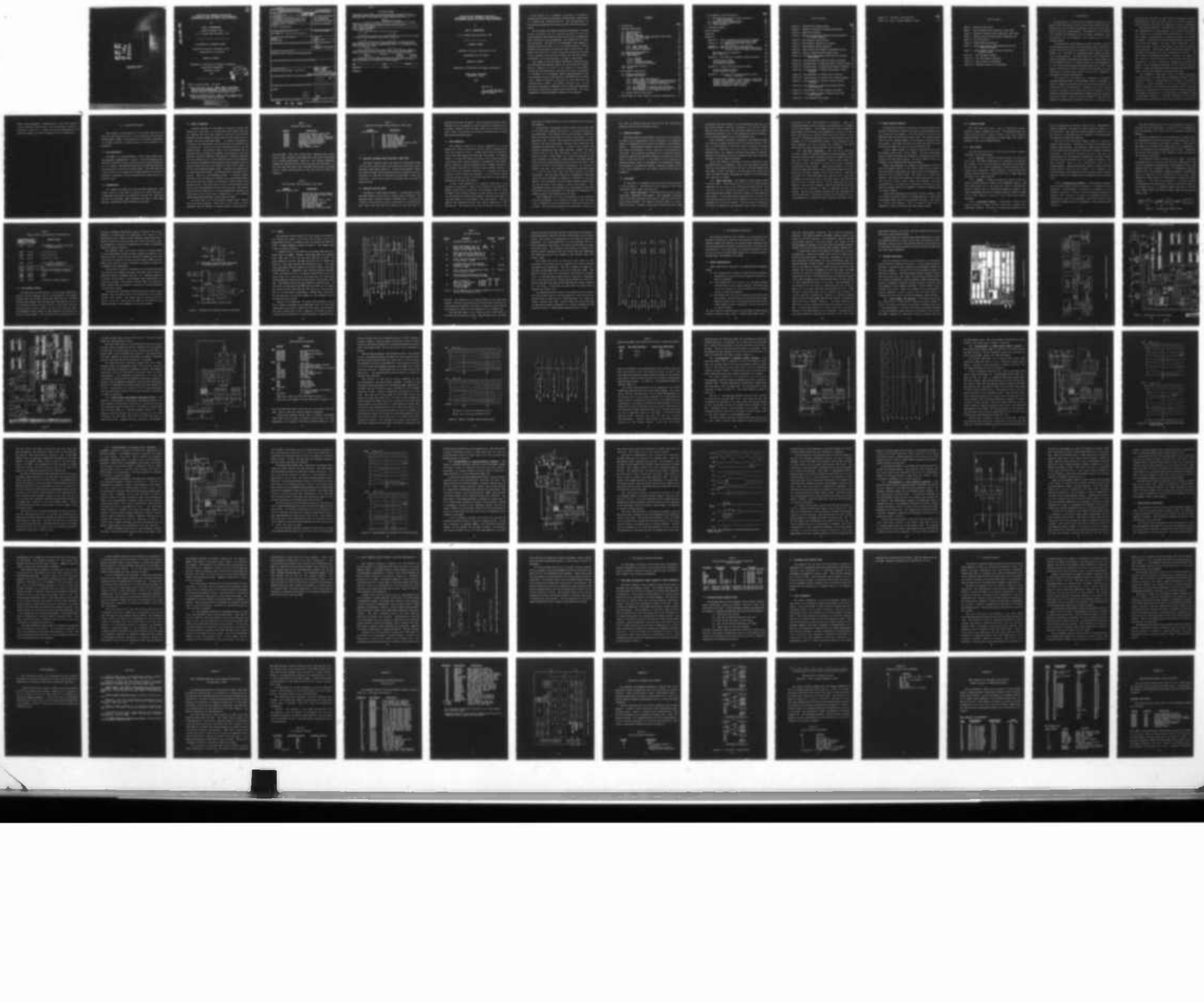
AD-A158 130 THE DESIGN AND HARDWARE EVALUATION OF AN ADVANCED 16-BIT LOW-POWER HIGH P..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH G S MAUERSBERGER 1985 1/2

UNCLASSIFIED AFIT/CI/NR-85-72T F/G 9/2 NL



1·0
2·8
2·5
3·15
2·2
3·5
1·1
2·0
4·0
4·5
1·8
1·25
1·4
1·6

1

# THE DESIGN AND HARDWARE EVALUATION OF AN ADVANCED 16-BIT, LOW-POWER, HIGH PERFORMANCE MICROCOMPUTER SYSTEM FOR DIGITAL SIGNAL PROCESSING

by

GARY S. MAUERSBERGER

Second Lieutenant, USAF

B. S., Kansas State University, 1983

---

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1985

---

Master's Thesis Length: 108 pages

Primary Information Source: AAMP, CAPS-7, and CAPS-10 Instruction Set Architecture, Processor Technology Section, Advanced Technology and Engineering, Collins Avionics Division, Rockwell International, 1982.

Secondary Information Source: Dave W. Best, "Design of an Advanced Architecture Microprocessor" (MS Thesis, Iowa State University, 1984), pp 100-114.

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER AFIT/CI/NR 85-72T | 2. GOVT ACCESSION NO. A158130 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) The Design and Hardware Evaluation of an Advanced 16-Bit, Low-Power, High Performance Microcomputer System for Digital Signal Processing | | 5. TYPE OF REPORT & PERIOD COVERED THESIS/~~DISSERTATION~~ |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Gary S. Mauersberger | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: Kansas State University | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433 | | 12. REPORT DATE 1985 |
| | | 13. NUMBER OF PAGES 188 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report) UNCLASS |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

Lynn Wolaver

LYNN E. WOLAVER
Dean for Research and
Professional Development
5 AUG 1985 AFIT, Wright-Patterson AFB OH

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

| Accession For | |
|---|---|
| NTIS GRA&I | ✓ |
| DTIC TAB | ☐ |
| Unannounced | ☐ |
| Justification | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist | Avail and/or Special |
| A-1 | |

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DDC QUALITY INSPECTED 1

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

85 8 13 082

85-72T

## AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (AU). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

RESEARCH TITLE: The Design and Hardware Evaluation of an Advanced 16-Bit, Low-Power, High Performance Microcomputer System for Digital Signal Processing

AUTHOR: Gary S. Mauersberger

RESEARCH ASSESSMENT QUESTIONS:

1. Did this research contribute to a current Air Force project?

( ) a. YES ( ) b. NO

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

( ) a. YES ( ) b. NO

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

( ) a. MAN-YEARS ________ ( ) b. $________

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3. above), what is your estimate of its significance?

( ) a. HIGHLY SIGNIFICANT ( ) b. SIGNIFICANT ( ) c. SLIGHTLY SIGNIFICANT ( ) d. OF NO SIGNIFICANCE

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the bottom part of this questionnaire for your statement(s).

| | | |
|---|---|---|
| NAME | GRADE | POSITION |
| ORGANIZATION | LOCATION | |

STATEMENT(s):

# THE DESIGN AND HARDWARE EVALUATION OF AN ADVANCED 16-BIT, LOW-POWER, HIGH PERFORMANCE MICROCOMPUTER SYSTEM FOR DIGITAL SIGNAL PROCESSING

by

GARY S. MAUERSBERGER

B. S., Kansas State University, 1983

---

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:

Donald H Lenhert

Major Professor

This thesis is a hardware evaluation of Rockwell International's Advanced Architecture Microprocessor (AAMP) which is a single-chip, CMOS/SOS device. The microprocessor is evaluated by designing and testing an AAMP based microcomputer system.

The Electrical and Computer Engineering Department at Kansas State University has an ongoing program to evaluate microprocessors and identify those with low power consumption, easy interfacing and programming, and capability of rapid multiplications. The AAMP is a microprocessor which appears to possess these qualities. It is a high-performance, 16-bit, CMOS/SOS microprocessor and operates from a 20 MHz clock. The AAMP's instruction set is well suited for compiling from high level languages, such as Jovial and Ada, and includes integer, fractional, and floating-point arithmetic operations.

This thesis covers the AAMP's operational characteristics, the developed microcomputer system, and the evaluation process. Special emphasis is given to the aspects of the AAMP's architecture and operational characteristics which are important in the design of a microcomputer system. The performance of the microcomputer system is determined by implementing the Widrow linear adaptive predictor algorithm in both fractional and floating-point formats. The AAMP's execution rate of the evaluation algorithm is compared to that of other microprocessors previously evaluated at Kansas State University. The power consumption of the microcomputer system is also measured.

CONTENTS

## List of Figures

## List of Tables

# 1. Introduction

The Department of Electrical and Computer Engineering at Kansas State University, under contract with Sandia National Laboratories, has an ongoing program to identify and evaluate "state of the art" microprocessors. The objective of the program is to identify microprocessors with low power consumption; easy interfacing and programming; and capability of rapid multiplications.

A microprocessor which appears to satisfy the above criteria is the Advanced Architecture Microprocessor (AAMP) designed by the Collins Avionics Group of Rockwell International at Cedar Rapids, Iowa. The AAMP is a high-performance, general-purpose, 16-bit microprocessor. The AAMP design was first implemented using two-micrometer CMOS/SOS VLSI technology.[1] The availability of the CMOS/SOS version is limited; however, an AAMP implemented in VLSI CMOS technology should become available in 1985. The general architecture of both versions will be the same, but some difference in timing and pin function does exist. Appendix A highlights the notable differences.

The purpose of this thesis is to evaluate the hardware oriented characteristics of the CMOS/SOS version of the AAMP. A Master's Thesis by Kenneth Albin evaluated the internal and instruction set architectures.[2] Kenneth Albin also encoded various versions of the Widrow and Lattice adaptive linear predictor algorithms and estimated the AAMP's execution times for these algorithms. There has been no attempt to duplicate the research accomplished by Kenneth Albin but to continue that evaluation by building an AAMP based microcomputer system, determining the actual execution rate for the Widrow adaptive linear predictor algorithm, and measuring the power consumption.

The features desired in the AAMP based microcomputer system were communication to a host computer via a parallel input/output port, execution of any AAMP encoded program sent to it from the host computer, and acquisition of analog information for digital processing. The Hewlett-Packard HP-9845B desktop computer was used as the host computer. The Datel Intersil ADC-HC12B, 12-bit analog-to-digital converter, was chosen for the analog-to-digital data acquisition. These were chosen because they satisfied the necessary requirements and were immediately available. The Harris 82C55A, programmable peripheral interface, was chosen for the parallel input/output port because of its speed and versatility. In order to execute any program sent from the host computer, the microcomputer system had to also include read-only-memory to house the operating system programs and random-access memory to store the programs for execution and data.

The remainder of this thesis covers the AAMP's operational characteristics, the developed microcomputer system, and the evaluation results. The description of the AAMP is contained in Section 2. Special emphasis is given to the aspects of the AAMP's architecture and operational characteristics which are important to the design of an AAMP based microcomputing system. The design and operation of the AAMP based microcomputer system is described in Section 3. Particular attention is given to the AAMP's timing characteristics and how they affect the choice of system timing parameters. Implementation of the Widrow adaptive linear predictor algorithm on the microcomputer system is discussed in Section 4. System performance measurements and concluding remarks are given in Sections 5 and 6 respectively.

# 2. Processor Description

The purpose of this section is to briefly describe the features and characteristic of the Advanced Architecture Microprocessor (AAMP). The objective is to provide a supplement to already available literature and is not intended to be exhaustive.

## 2.1 Chip Description

The AAMP is a high-performance, 16-bit, stack architecture microprocessor implemented on a single, silicon-on-sapphire die. The AAMP is housed in 68 pin, square pin-grid package measuring 1.1 x 1.1 inches. The CMOS/SOS technology provides the AAMP with the qualities of low power and high speed. The AAMP operates from a single +5 volt supply and consumes between 50 and 125 mW of power.[1] The AAMP is rated to operate with a 20 MHz crystal or external clock.

## 2.2 Compatibility

All inputs and outputs are TTL and CMOS compatible. The output fanout is one when interfacing to TTL; therefore, transceivers and buffers are required to isolate the AAMP from the rest of the system. When interfacing the AAMP to other CMOS devices, conventional CMOS interfacing rules apply.

## 2.3 Modes of Operation

Programs executed in the AAMP can reside in either the executive or user mode. In a system where multitask functions are needed, the user mode should be used for the working programs, and the executive mode should be the "executive" or manager over the various user tasks. If, however, the system is to execute only one program, that program can be defined as the initialization routine and can be operated entirely in the executive mode. The executive mode is the "privileged" mode of operation and is intended for processor initialization, bus error handling, interrupt handling, trap handling, exception handling, and task scheduling. All these functions need not be defined. The processor determines which executive mode programs exist by address offsets given in the Executive Entry Table (Table 1). The first nine read-only memory (ROM) locations are reserved for the Executive Entry Table. An address offset referred to as a pointer (PTR) is a word address offset. An address offset referred to as a procedure ID (PROCID) is a byte address offset. The equivalent word address is obtained by right shifting the PROCID one bit. The right most bit tells the processor which byte of the corresponding word starts the program. The program starts with the high byte or low byte depending on whether the right most bit is a one or zero, respectively. A zero address offset means that particular routine does not exist.

The user mode is for the general purpose "user" defined functions and programs. There can be any number of user programs. They are defined in a manner similar to the executive

Table 1

Executive Entry Table

| Address | Description |
|---|---|
| 000000 | Continuation Status Pointer (PTR) |
| 000001 | Initial Exec. Stack Limit PTR |
| 000002 | Initial Exec. Top of Stack (TOS) PTR |
| 000003 | Initialization Procedure ID (PROCID) |
| 000004 | Bus Error Interrupt PROCID |
| 000005 | Nonmaskable Interrupt PROCID |
| 000006 | Maskable Interrupt PROCID |
| 000007 | Trap PROCID |
| 000008 | Exec. Exception PROCID |

mode programs. Each user program has a separate entry table called a User Processor State Descriptor (PSD) Table which must lie in random access memory (Table 2). The primary difference between the Executive Entry Table and the User PSD Table is that the User PSD Table also contains the syllable program counter (SPCR) which points to the next byte of code for execution.

Table 2

User Processor State Descriptor (PSD) Table

| Address (User PSD Pointer plus) | Description |
|---|---|
| 0 | Initial User Stack Limit Pointer |
| 1 | Initial User Top of Stack Pointer |
| 2 | Local Environment |
| 3 | Data Environment |
| 4 | Syllable Program Counter (SPCR) |
| 5 | Code Environment |
| 6 | Task Procedure ID (PROCID) |
| 7 | Task Code Environment |
| 8 | User Exception PROCID |
| 9 | User Exception Code Environment |

Table 3

Executive Processor State Descriptor (PSD) Table

| Address (Initial TOS plus) | Description |
|---|---|
| 0 | User PSD Pointer |
| 1 | Exec. Stack Limit (SKLM) |
| 2 | Exec. Top of Stack (TOS) |
| 3 | Exec. Local Environment |
| 4 | Exec. Data Environment |
| 5 | Exec. Syllable Program Counter (SPCR) |
| 6 | Exec. Code Environment |
| 7 | Interrupt Enable Flip-Flop |
| 8 | Exec. Error Code |

## 2.4 Executive Processor State Descriptor (PSD) Table

The eight locations above the executive top-of-stack (TOS) location are reserved for the Executive Processor State Descriptor (PSD) Table (Table 3). Anytime the processor leaves the executive mode, the processor's state, including any error condition, is recorded in the Executive PSD Table. Thus, the information can be very helpful in determining the cause of system failure.

## 2.5 Executive and User Stack

The AAMP is a stack-type processor. A cache memory inside the AAMP, makes up the top four usable stack locations. The remaining stack locations reside in external RAM. The size and location of the executive stack is specified in the Executive Entry Table and the size and location of the user stack is defined in the User PSD Table. Only one executive stack can be defined, but it is possible to have as many user stacks as user programs. However, only one user stack can be active at any given time. The active user program is specified by the User PSD Pointer described below.

## 2.6 *Task Scheduling*

The executive mode is by design responsible for task scheduling. The initialization routine usually schedules the first user program or task. A context switch from the executive to the user mode requires that the User PSD Pointer, the address offset to the User PSD Table, be stored in the initial executive top-of-stack (TOS) location defined in the Executive Entry Table. When an executive program terminates with a RETURN instruction, the processor does what is called an "outer procedure return", a return from the outermost procedure of a program. The outer procedure return causes the processor to do a context switch from the executive mode to the user mode and run the program specified by the User PSD Pointer.

Further task scheduling is normally performed by the executive trap handler. When a user program is finished and a RETURN instruction is executed, the processor does an outer procedure return which restores the User PSD to its original values and forces a context switch back to the executive mode. A trap code number 0 is also passed to the trap handler. The trap handler then schedules the next user task by updating the User PSD Pointer. The context switch from the executive mode to the user mode is accomplished by an outer procedure return from the trap handler.

The executive trap handler can be invoked from the user mode in two other ways besides the normal user program termination just described. The context switch from the user to executive mode can also be initiated by either a processor or user generated trap. Illegal instructions encountered in the user program, user stack overflows, and user stack underflows are considered nonrecoverable errors. If such an error occurs, the processor suspends the user task's execution, updates the task's process state in the User PSD Table, places on top of the executive stack a trap number (0-7) which corresponds to the error condition, and forces a context switch to the trap handler. Traps can also be generated from within the user program. Trap numbers (8-65,535) are software generated. To initiate a software trap, the user program must place the trap number minus 8 on top of the user stack and then execute the TRAP instruction. The user task's execution is suspended as described above and the trap number is passed as a parameter to the trap handler. The trap handler can then either schedule another task, correct the error if one exists, or halt the processor. If no trap handler exists, the processor will stop execution and store an executive error code number 13 in the Executive PSD Table. To resume execution, the processor must be reset.

Task scheduling can also be performed by the executive interrupt procedures. If an external interrupt occurs, the user task is suspended and the associated interrupt procedure is invoked. The interrupt procedure can then define a new active user task by changing the User PSD Pointer and terminating execution with an outer procedure return.

## 2.7 Exception Handling

When the result of an arithmetic operation is too large or too small to be represented in the accumulator, an exception number corresponding to the problem is passed to the exception handler. The exception handler must exist in the processor mode where the exception occurs, such as in the executive or user mode. The processor determines if an exception handler exists by checking the exception PROCID in the Executive Entry Table or User PSD Table. If the PROCID is zero (meaning no exception handler exits), the processor will automatically handle the exception by placing the largest or smallest possible value on the stack depending on the number format and the type of exception.

## 2.8 Interrupts

An interrupt is a request for service from a device external to the processor. The AAMP has three types of interrupt structures: event, reset, and bus error.

2.8.1 Event Interrupts -- The AAMP has two forms of event interrupts: nonmaskable and maskable. The nonmaskable interrupt (NMI) is an edge-triggered input which is ideally suited for a power failure system. The maskable interrupt (INT) is a level-sensitive input. Maskable interrupts are enabled by the INTE instruction when the processor is in the executive mode and are are always enabled when the processor is in the user mode. When an interrupt occurs, the processor completes the current instruction and does a call to the appropriate interrupt handler. If the interrupt PROCID is zero the processor halts.

Once a maskable interrupt is enabled in the executive mode, the interrupt remains enabled until an interrupt occurs. The call to the maskable interrupt handler disables further interrupts; therefore, to allow later interrupts in the executive mode, the maskable interrupts must again be enabled. Maskable interrupts can also be enabled within the interrupt procedure to allow nested interrupts.

Since the maskable interrupt is level sensitive, some precautions must be considered. First, the logic high to the INT input pin must remain high until the call to the INT handler has been made. Second, the input to the INT input must return to a logic low before further interrupts are enabled, or the interrupt will nest back on itself.

2.8.2 Reset Interrupts -- The AAMP has a dual-purpose reset interrupt (RST). The processor will do a cold or warm restart depending on the value of the continuation-status word. The location of the continuation-status is defined by the continuation-status pointer, the first location in the Executive Entry Table. If the continuation-status pointer is zero, the continuation-status word is also defined to be zero, and the processor will do a cold restart. The cold restart configures the stack as defined in the Executive Entry Table and starts execution of the initialization procedure. When the continuation-status word is nonzero, the processor will attempt to do a warm restart. If the executive error code in Executive Entry Table is also nonzero, the processor will return to an idle state and wait for a second reset which will force a cold restart.[3] The nonzero executive error code indicates that the processor is in a nonrecoverable state. A zero executive error code indicates that the processor was stopped by the HALT instruction and that program operation can be resumed.

2.8.3 Bus Error Interrupt -- The bus error interrupt provides a means for external hardware to provide memory access protection and processor recovery from data transfer failures such as a device not responding.[4] The bus error signal is returned to the processor in place of a normal transfer acknowledge (XAK) signal. When the processor is in the executive mode, the bus error will cause the processor to halt. Operation can only continue with a cold restart. When the processor is in the user mode, the bus error will cause the processor to store the state of the user task in the User PSD Table and do a context switch to the executive bus error handler. The bus error handler then determines what program execution is to continue, if any, by updating the User PSD Pointer. The bus error handler can also halt the processor with a HALT instruction. Note that if the bus error PROCID is zero, the processor will always halt when a bus error interrupt occurs.

## 2.9 Memory Address Formation

The AAMP has 24 address lines and two status lines which can be used to effectively address 32 M bytes of program memory and 32 M words of data memory. The two status lines are the code/data ($C/\overline{D}$) and the executive/user ($E/\overline{U}$). The executive/user line allows for external memory to be divided into executive and user spaces. Each space can then be further divided into code and data environments by the use of the code/data line.

A 24-bit memory address is formed by concatenating an 8-bit environment pointer with a 16-bit offset. The 8-bit environment pointer allows 256 unique environments to be defined in both the executive and user memory spaces. The 16-bit offset allows 64 K elements to be defined within each environment.

Memory code addresses are formed by concatenating the 8-bit code environment pointer to the left of the 16-bit syllable program counter (SPCR) which is a byte offset corresponding to the next byte of program code. The result is a 24-bit byte address. The 24-bit byte address is right shifted one bit and zero filled to form a 24-bit memory word address. The most significant bit is always zero; therefore, 8 M words or 16 M bytes of program code can be accessed in both the executive and user memory spaces.

Memory data addresses are formed in a similar manner. The 8-bit data environment pointer is concatenated with a 16-bit data environment offset given by the program's execution. The result is a 24-bit word address. Thus 16 M words of data can be addressed in both the executive and user memory spaces.

## 2.10 Addressing Modes

The AAMP has four basic types of addressing modes: universal, global, indexed, and local. The addressing modes can access single (16-bit), double (32-bit), and triple (48-bit) precision words. For a description of these modes, the reader is referred to the work by Kenneth Albin.[2]

## 2.11 Data Formats

The AAMP can work with four data types: Boolean, integer, fractional, and floating-point.

2.11.1 Boolean -- Boolean variables require the use of a 16-bit word. The least significant bit is defined as the Boolean bit; however, all 16 bits are used by instructions which require Boolean data. All zeros represents a FALSE condition and a nonzero word represents a TRUE condition. Instructions that generate Boolean data will produce "0000" for a FALSE condition and "0001" for a TRUE condition.

2.11.2 Integer -- Integer variables are single (16-bit) or double (32-bit) precision, signed, two's complement numbers. The most significant bit is the sign bit with "0" representing positive numbers and "1" representing negative numbers. The least significant portion of a double precision integer resides in the lower address location of two consecutive memory addresses.

2.11.3 Fractional Numbers -- Fractional variables are single (16-bit) or double (32-bit) precision, signed, two's complement numbers. The sign bit is the most significant bit with "0" representing a positive fraction and "1" representing a negative fraction. The range for single precision numbers is -1 to (1-2**(-15)). The range for double precision numbers is -1 to (1-2**(-31)).[3] The binary point is assumed to be fixed between the sign bit and the first data bit.

The use of the fractional number format is limited to special applications. Accumulator overflows and underflows present a major problem. The fractional number format works well if the multiplication operation is the only arithmetic operation performed. The AAMP is designed in such a way that the multiplication of two fractions conforming to the above format, one sign bit and 15 fraction bits, will produce a product of the same format. This is not necessarily true on all microprocessors. Some microprocessors return a product where the binary point moves to the right.[5] For example, the multiplication of two fractions with one sign bit and 15 fraction bits will result in a product with two sign bits and 14 fraction bits.

2.11.4 Floating-Point Numbers -- Floating-point variables can be either single (32-bit) precision or extended (48-bit) precision. Floating-point variables are composed of three fields: a sign bit; a 24-bit or 40-bit mantissa depending on whether the variable is single or extended precision, respectively; and a 8-bit exponent. Note that one bit of the mantissa is hidden and is not considered in the variable length (Figure 1).

The most significant bit of the floating-point variable is the sign bit for the mantissa. A sign bit of "0" represents a positive mantissa, and a sign bit of "1" represents a negative mantissa.

The most significant bit of the mantissa is hidden and is not represented in the actual variable. The binary point is to the left of the hidden bit as shown in Figure 1. The value of the hidden bit is always "one" except when zero is expressed. A zero is represented when the exponent field is zero. Regardless the value in the mantissa, if the exponent is zero, the value of the variable is also zero.

The exponent field is eights bits in length. The most significant bit is the sign bit and the seven remaining bits are the two's complement representation of the exponents value. The exponent sign bit is inverted so that a "1" represents a positive exponent and a "0" represents a negative exponent. As explained above, an exponent of zero defines the value "zero". Some examples of single precision floating-point variables are given in Table 4. Note that the exponent is a straight binary representation from the smallest positive (or negative) number to the largest positive (or negative) number with zero as the only valid number between the positive number range and the negative number range.

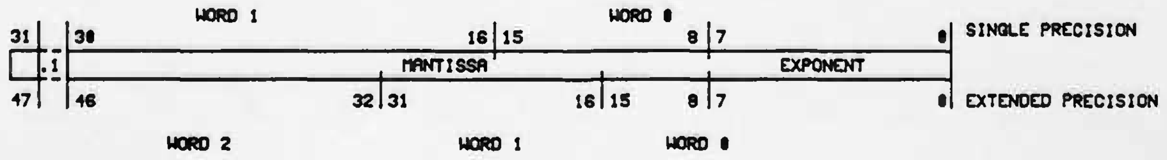


Figure 1. Floating-Point Number Format

Table 4

Single Precision Floating-Point Representation

| Hexadecimal Value | | | Decimal Value |
|---|---|---|---|
| Word 1 | Word 0 | | |
| Mantissa | | EXP | |
| 7FFF | FF | FF | ≅ $0.9999999702 \times 2^{127}$ ≅ 1.7014 E38 (largest positive) |
| 0000 | 00 | 81 | = $0.5 \times 2^{1}$ = 1.0 |
| 0000 | 00 | 80 | = $0.5 \times 2^{0}$ = 0.5 |
| 0000 | 00 | 7F | = $0.5 \times 2^{-1}$ = 0.25 |
| 0000 | 00 | 01 | = $0.5 \times 2^{-127}$ = 2.9387 E-39 (smallest positive) |
| XXXX | XX | 00 | 0.0 |
| 8000 | 00 | 01 | -2.9387 E-39 (smallest negative) |
| 8000 | 00 | 7F | -0.25 |
| 8000 | 00 | 80 | -0.5 |
| 8000 | 00 | 81 | -1.0 |
| FFFF | FF | FF | -1.7014 E38 (largest negative) |

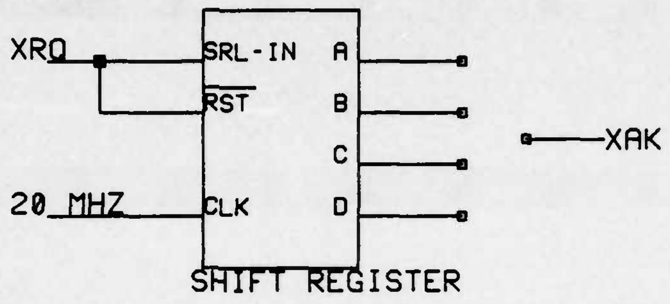
## 2.12 Bus Transfer Protocol

The AAMP uses an asynchronous, interlocking, handshaking protocol for bus transfers. This is accomplished by the transfer request (XRQ) and transfer acknowledge (XAK) signals. The AAMP initiates the transfer with the XRQ signal. The transfer cycle is not terminated until the AAMP receives a XAK signal from the external device being accessed or a bus error interrupt (XER). The handshake protocol makes the AAMP compatible with devices of widely varying speeds. Most devices such as memories do not provide a transfer acknowledge signal; therefore, the circuit designer is responsible for generating this signal at the appropriate time. Matters become somewhat complicated when a system is made up of devices that have differing access times. Two examples of circuits to generate the transfer acknowledge signal are shown in Figure 2.

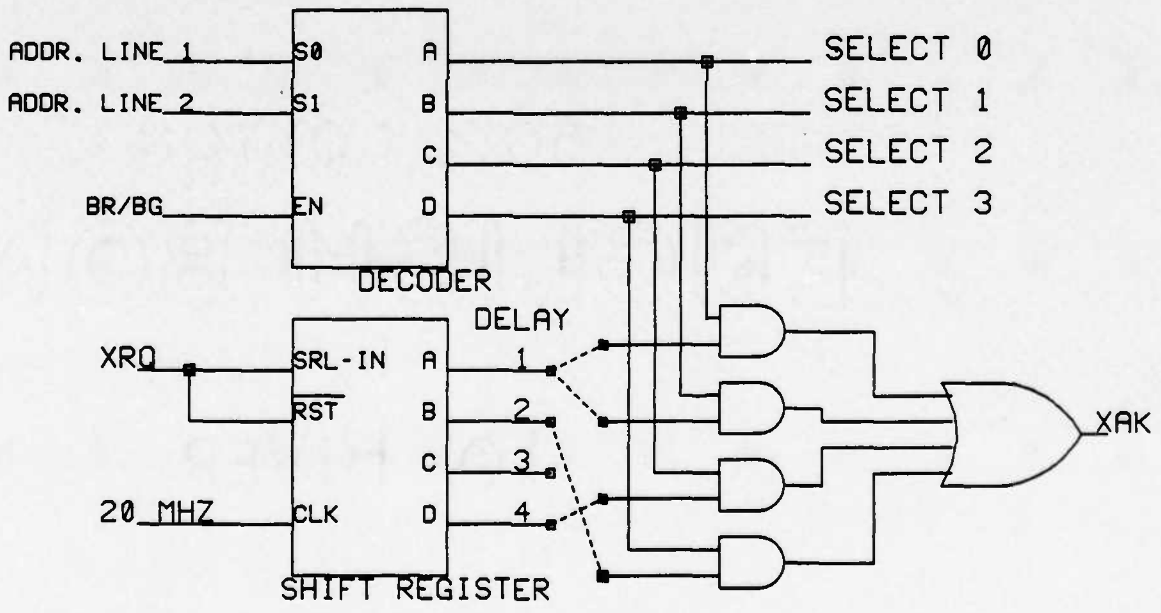
The address, data, and status lines of the AAMP become invalid at the same time that transfer request is removed. If an external device or bus protocol requires an address or data hold time after the negation of XRQ, the hold output ($\overline{\text{HLD}}$) from the AAMP can be used to control external transparent latches to accommodate this requirement.

The read/write output of the AAMP is valid during the same time period as the address and data lines and can't be used as a read/write pulse to external devices. Its intended purpose is to control the direction of the external data transceivers in a common bus system. The transfer request (XRQ) and read/write ($\overline{\text{R}}$/W) signals must be combined externally to generate memory read and write pulses.

The AAMP has a selectable setup time between enabling the address, data, and status lines to the assertion of the transfer request (XRQ) signal. The setup time is controlled by two inputs, S0 and S1. This gives the AAMP four different setup delays and makes it more adaptive to any given system.



(a) Fixed delay of XAK wrt XRQ (determined by slowest device in the circuit)



(b) Selectable delay of XAK wrt XRQ

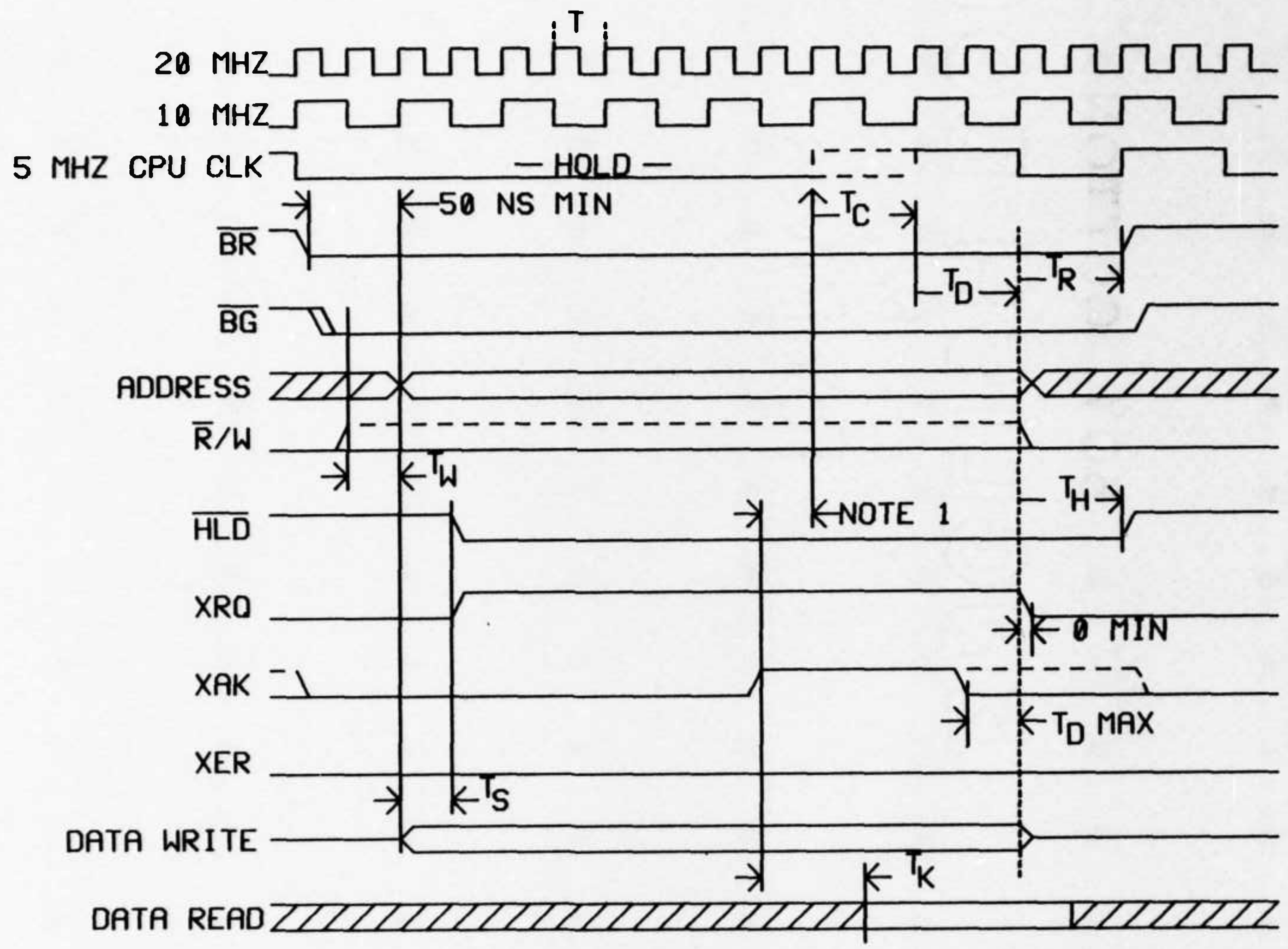
Figure 2. Methods of Generating Transfer Acknowledge

## 2.13 Timing

The external 20-MHz crystal or clock signal is divided by two inside the AAMP to provide the 10-MHz clock for synchronization and is divided by four to provide the 5-MHz CPU clock. A timing diagram is shown in Figure 3. The nominal time values are shown in Table 5.

A transfer cycle is initiated by bus request ($\overline{BR}$) going low. Before this can happen, transfer acknowledge (XAK) and transfer error (XER) must be low and bus grant ($\overline{BG}$) must be high. If these conditions are met, the AAMP initiates the bus request and waits for a bus grant from the bus arbitration logic. On a read cycle, the setup time ($T_S$) begins at the next 20-MHz clock edge after $\overline{BG}$ goes low. Therefore, even with $\overline{BG}$ connected directly to $\overline{BR}$, one period (50 ns) is required between $\overline{BR}$ low and the start of $T_S$. On a write cycle, the read/write line ($\overline{R}$/W) goes high when $\overline{BG}$ goes low which starts the delay to output data valid ($T_W$). The address setup time ($T_S$) begins at the end of $T_W$. If the system is configured with $\overline{BG}$ connected to $\overline{BR}$, $T_S$ will begin one period (50 ns) after $\overline{BR}$ goes low since $T_W$ is one period. Thus in a no wait state configuration, address setup time occurs at the same point in time with respect to bus request for both the read and write cycles.

During a write operation, the setup time selected by the S0 and S1 inputs is extended by one additional cycle to allow data transceivers extra time to switch from the normal read direction. (Note that transceivers are only required if the AAMP is used in a system with a shared bus or when the system is composed of TTL



NOTE 1 -- 0 TO 100 NS PLUS XAK PROPAGATION TIME INSIDE AAMP

Figure 3. AAMP Timing Diagram

TABLE 5

Bus Timing Values

| Symbol | Parameter | | Minimum | Maximum |
|---|---|---|---|---|
| $T$ | Oscillator period at input Y0 | | 50ns | |
| $T_C$ | Delay time between sync. of XAK to the 10-MHz clock and restart of the 5-MHz CPU clk. | READ | 2T | |
| | | WRITE | 0 | |
| $T_D$ | Delay time from the restarting of the 5-MHz CPU clock to XRQ low or any change in A23-A00, R/W, D15-D00 | | 2T | |
| $T_H$ | Delay time from the negation of XRQ to the negation of $\overline{HLD}$. | | 2T | 2T |
| $T_K$ | Allowable skew from the received assertion of XAK to the validity of D15-D00 as input. | | | 4T-150 |
| $T_R$ | Delay time from the negation of XRQ to the negation of $\overline{BR}$. | | 2T | (note 1) |
| $T_W$ | Time from the assertion of $\overline{R}$/W high to the enabling of D15-D00 as output. | | T | |
| | | | (READ) | (WRITE) |
| $T_S$ | Setup from $\overline{BG}$ low, XAK low, A23-A00 valid, $\overline{R}$/W valid, D00-D15 valid to assertion of XRQ and $\overline{HLD}$. | S1,S0=00 | T | 2T |
| | | S1,S0=01 | 2T | 3T |
| | | S1,S0=10 | 3T | 4T |
| | | S1,S0=11 | 4T | 5T |

(note 1: $T_R$ is normally 2T if no data transfer is to occur on the following machine cycle.)

devices.) The read/write line always returns to the read state between transfers. The majority of code and data transfers are read operations; therefore, the choice to have the read direction the quiescent direction of the transceivers is an optimal one.

Once XRQ goes high, it will remain so until a XAK signal is received from the external device. When XAK goes high, the delay time $T_C$ for restarting the 5-MHz CPU clock begins at the point of synchronization of XAK and the next rising edge of the 10-MHz clock. The time $T_C$ is dependent on the type of transfer operation. On a read operation, $T_C$ is two periods; but on a write operation, $T_C$ is zero. The extra two periods on a read operation allow for internal chip delays in latching the received data.[6] The delay time ($T_D$) is the time from restarting the 5-MHz CPU clock to any changes of the address, data, and control lines.

The 5-MHz CPU clock is halted at the center of the microcycle following the bus request and remains halted throughout the transaction. The bus transfer is ended by a transfer acknowledge (XAK) or bus error interrupt (XER). The AAMP can retain control of the bus for more than one transfer cycle. If the AAMP does retain control of the bus a subsequent cycle, the setup time will begin and the 5-MHz CPU clock will halt two periods after XRQ low provided XAK has already returned to a low state. If XAK goes low after XRQ, the setup time is delayed until XAK is low.

The synchronization of XAK to the 10-MHz clock, the extra one period setup time for a write operation, and the extra two periods delay of XRQ low for a read operation, all affect the overall transfer cycle time. Figure 4 shows the timing waveforms for XRQ when the AAMP is operating in a no-wait configuration, $\overline{BR}$ connected directly to $\overline{BG}$ and XRQ connected directly to XAK.

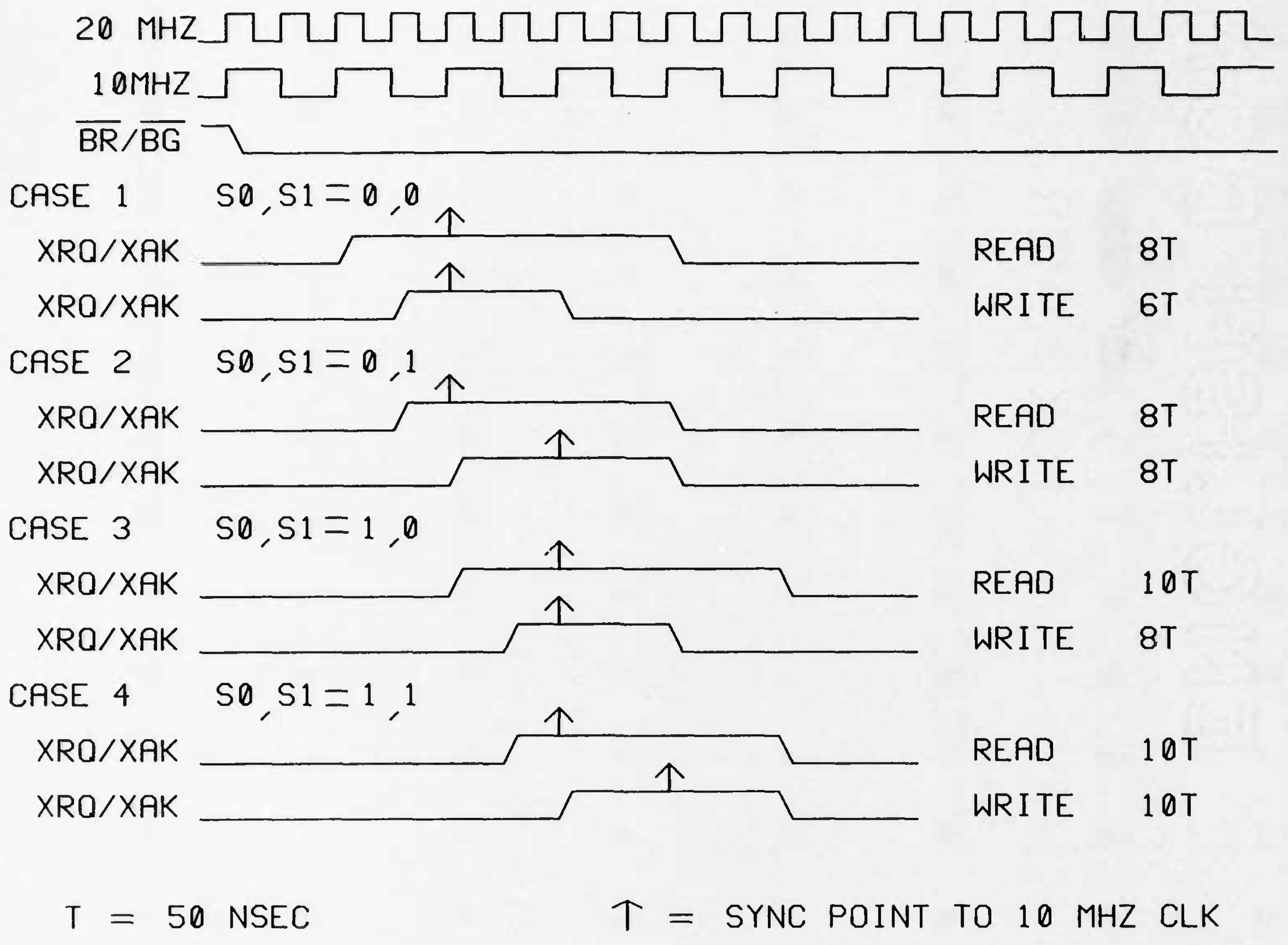


**Figure 4. Effects of Selectable Setup and Synchronization on Transfer Request**

# 3. Microcomputer Description

The primary purpose of this thesis is to incorporate the Advanced Architecture CMOS/SOS Microprocessor (AAMP) into a computing system and identify any problem areas associated with the microprocessor's operating characteristics and available documentation. To my knowledge, the system described in this section is the first AAMP based system built outside Rockwell International, the designer of the AAMP.

## 3.1 Design Considerations

Three features were required in the AAMP based microcomputer system.

(1) The system had to be able to send and receive digital information to and from a host computer by means of a parallel interface.

(2) The system had to be able to run any AAMP encoded program sent to it by a host computer. The system was not to be limited to just the execution of programs stored in read-only memories.

(3) The system had to be able to process both digital and analog data. The motivation for having analog signal sampling capability is so the system can be used to test linear adaptive filter algorithms, such as the Widrow, on real-time data.

The above features required that the microcomputer system design include a read-only memory (ROM), a random access memory (RAM), a parallel input/output interface, and an analog-to-digital converter. The Hewlett-Packard, HP-9845B, desktop computer was chosen as the host computer. The HP-98032A, general purpose input/output parallel interface (GPIO), was chosen as the Hewlett-Packard interface to be used with the HP-9845B computer because of its 16-bit unidirectional data transfer capabilities.

To effectively realize the AAMP's high speed and low power consumption, a system was needed which would not appreciably degrade these attributes. Therefore, the maximal use of high speed CMOS devices was chosen over the traditionally used low-powered Schottky (LS-TTL) devices. The high speed CMOS devices, such as the 54HC/74 HC family of CMOS circuits, have the noise immunity and low power consumption of the older metal gate CMOS devices but operate at speeds similar to the LS-TTL family of integrated circuits. The 54HC/74HC family of CMOS devices has lower power consumption than the 54LS/74LS family of TTL devices at input frequencies below 1 to 10 MHz. The difference is attributed to the fact that all LS-TTL devices use a constant amount of dc current even when the devices are idle. CMOS devices, on the other hand, have an almost zero quiescent dc current and power consumption. However, at operating speeds above 10 MHz, the high speed CMOS devices actually consume more power than the low-power Schottky. One might question the decision to use all CMOS devices since the system clock is 20 MHz. A close examination will reveal that only a few gates external to the AAMP operate at this frequency. The majority of the gates operate at frequencies far below 1 MHz; thus, high speed CMOS provides a more power efficient system than could be obtained by using LS-TTL devices.

A second reason for choosing high speed CMOS devices is that the CMOS/SOS AAMP has a fanout of one when interfacing to TTL. Therefore, the use of all CMOS devices reduces the chip count of the system by removing the need of buffers between the AAMP and the other system components.

## 3.2 Hardware Description

The microcomputer system is constructed on a perforated board using wire wrapping techniques. A 50-pin edge connector provides a connection point between the microcomputer system and the host computer. A picture of the microcomputer system is provided in Figure 5. The microcomputer system consists of seven basic sections: the clock, reset, and interrupt; device select and transfer acknowledge; read-only memory; random access memory; parallel peripheral interface; analog-to-digital converter; and microcomputer to host computer interface. Figure 6 shows a block diagram and Figure 7 shows the schematic for microcomputer system. Appendix B gives a parts list and board layout. A discussion of each section of the microcomputer system follows.

3.2.1 Clock, Reset, and Interrupt -- The timing source for the AAMP can be a 20 MHz crystal connected directly between Y0 and Y1 or 20 MHz clock signal clock signal connected to Y0 with Y1 left disconnected. A 20 MHz clock signal was needed elsewhere in the microcomputer system design, so an external clock circuit was added. Resistor R1, capacitors C1 and C2, inverter U1A, and

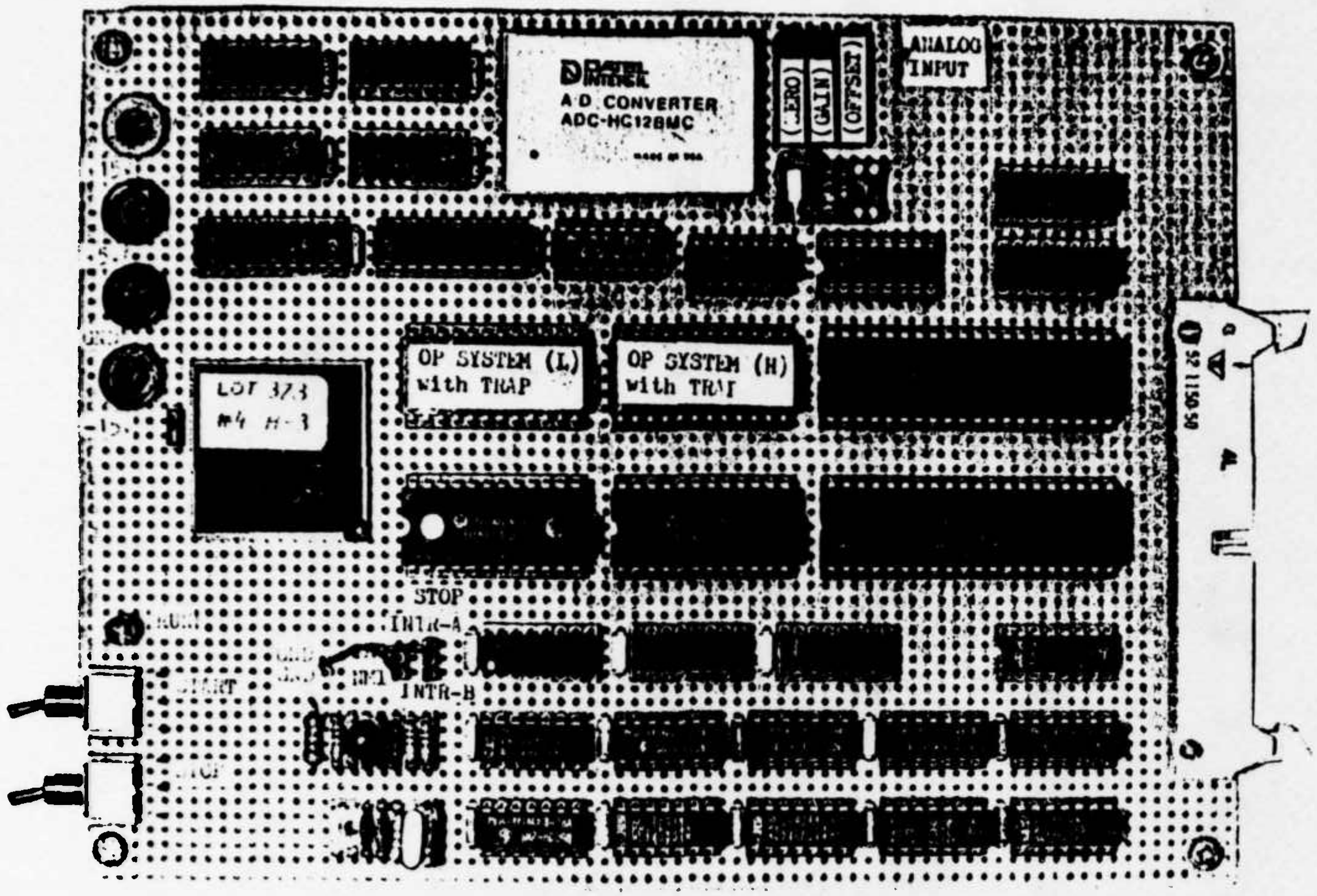


Figure 5. Microcomputer System Picture

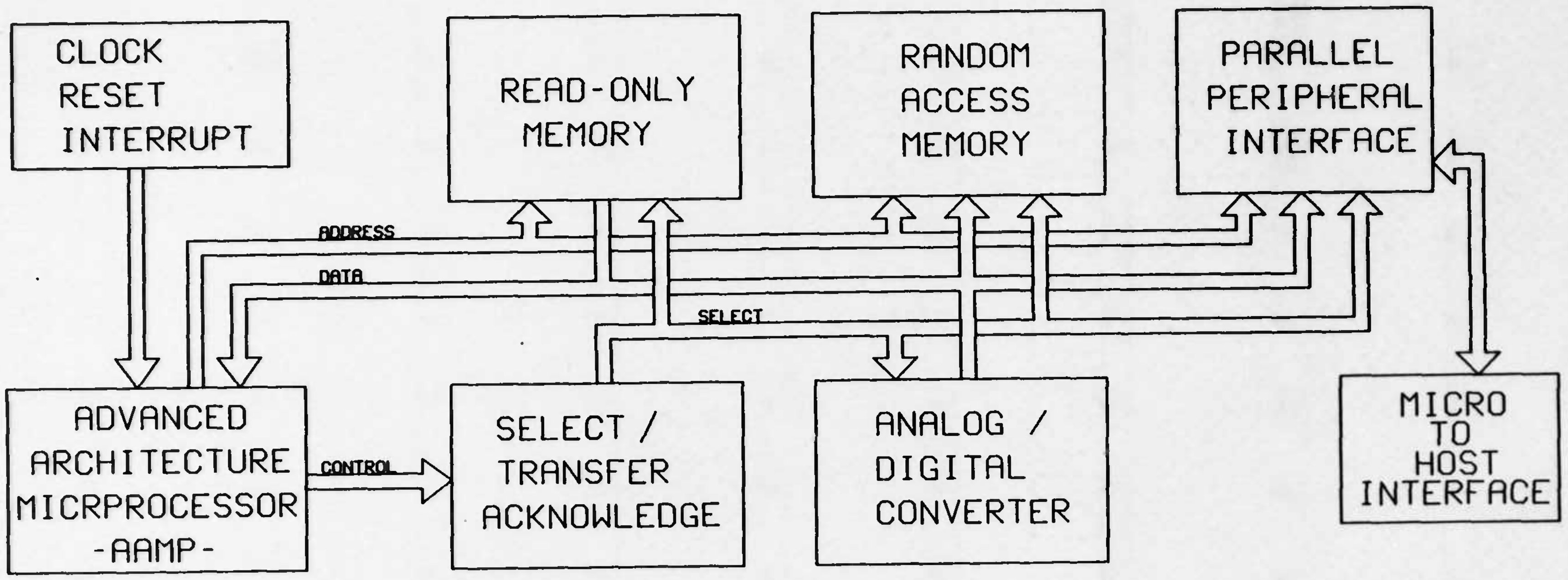


Figure 6.  Microcomputer System Block Diagram

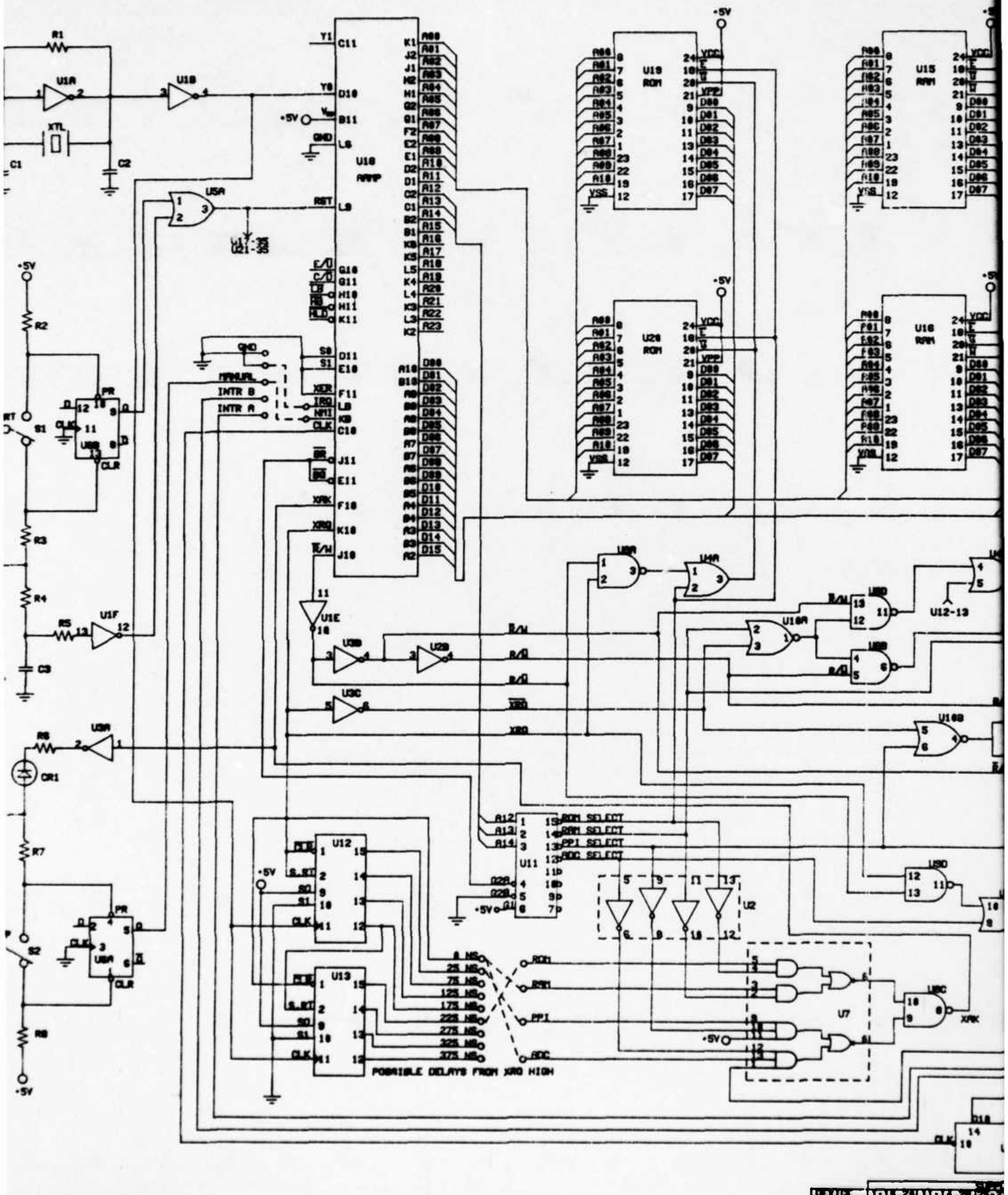
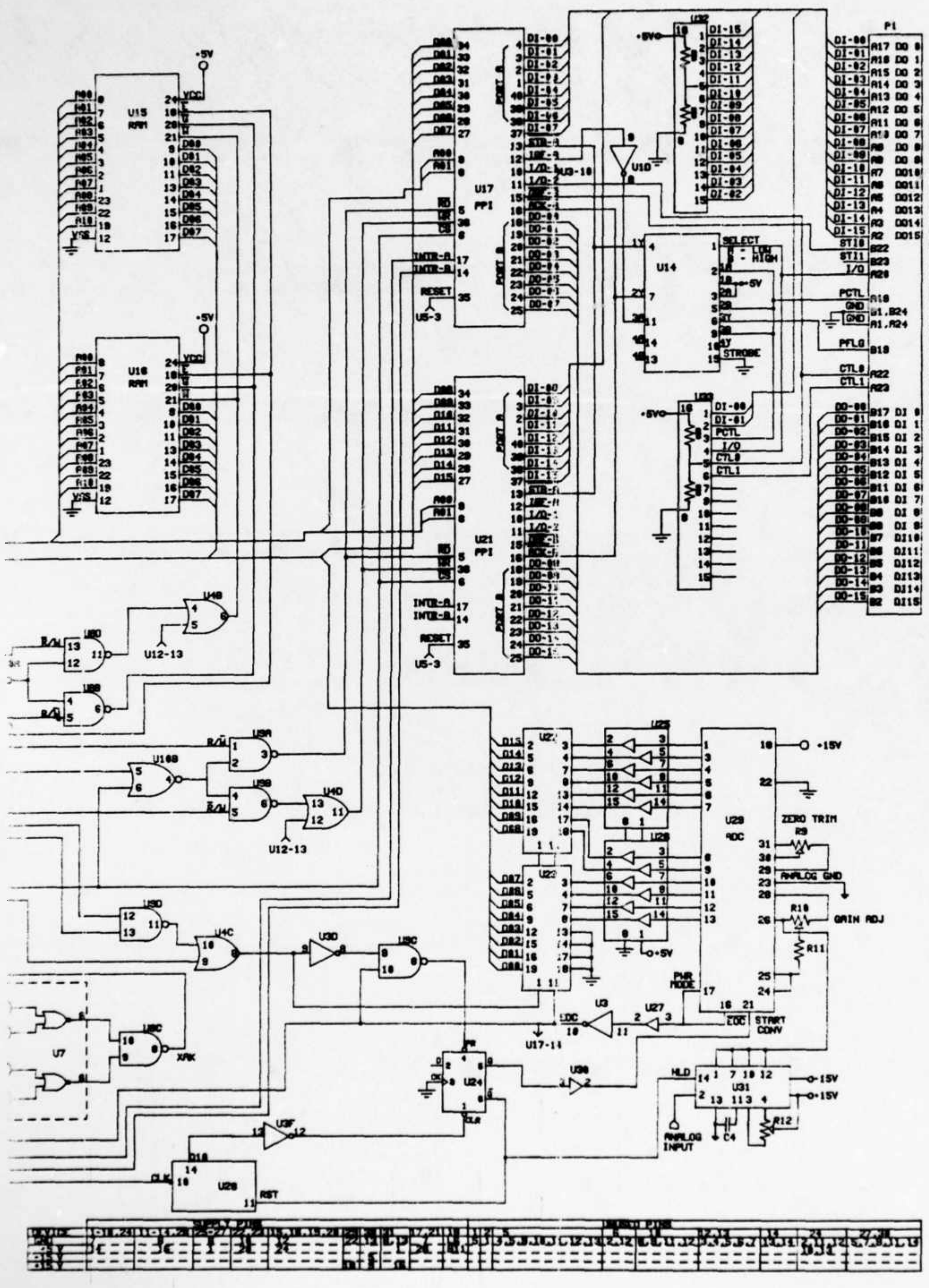


Figure 7. Microcomputer System Schematic

1/2


U15 RAM
U16 RAM
U17 PPI
U21 PPI
U14
U29 ADC
U31
U24
U28
U7
P1
PORT A
PORT B
SELECT
STROBE
ZERO TRIM
GAIN ADJ
ANALOG GND
ANALOG INPUT
PWR MODE
START CONV
XAK
RESET
U5-3
U12-13
U17-14
+5V
+15V
-15V
TO HP-98032A GENERAL PURPOSE INPUT/OUTPUT INTERFACE

2Ø2

a 20 MHz crystal make up the clock circuit. U1B is a output buffer for the oscillator circuit.

The reset (RST) signal to the AAMP comes from two sources. On power turn-on, resistor R4 and R5, capacitor C3, and inverter U1F generate a 260 ms positive pulse through OR gate U5A. A manual reset is also provided with switch S1. The D-type flip-flop U6B, resistors R2 and R3, and switch S1 produce a debounced positive pulse through OR gate U5A when S1 is toggled.

The AAMP has three interrupt inputs: bus error (XER), maskable (IRQ), and nonmaskable (NMI). The bus error interrupt is not used and is permanently connected to ground. The maskable interrupt (IRQ) and nonmaskable interrupt (NMI) are user configurable. The suggested configuration is to have NMI connected to switch S2 and the maskable interrupt connected to ground or the programmable peripheral interface (PPI) interrupts. Switch S2 is debounced by D-type flip-flop U6A and resistors R7 and R8. Connecting the Q output of U6A to the nonmaskable interrupt provides a means of halting the processor regardless its mode of operation.

3.2.2 <u>Device Select and Transfer Acknowledge Circuit</u> -- The AAMP has an asynchronous, interlocking, handshaking protocol for bus transfers as described in Section 2.12. Since the bus of the microcomputer system is dedicated to the AAMP, bus request ($\overline{BR}$) and bus grant ($\overline{BG}$) are connected together (Figure 8). Only address lines $A_0$ through $A_{10}$ and $A_{12}$ through $A_{14}$ are used; therefore, the code, data, and local environment pointers are restricted to zero. Address lines $A_{12}$ through $A_{14}$ are decoded by

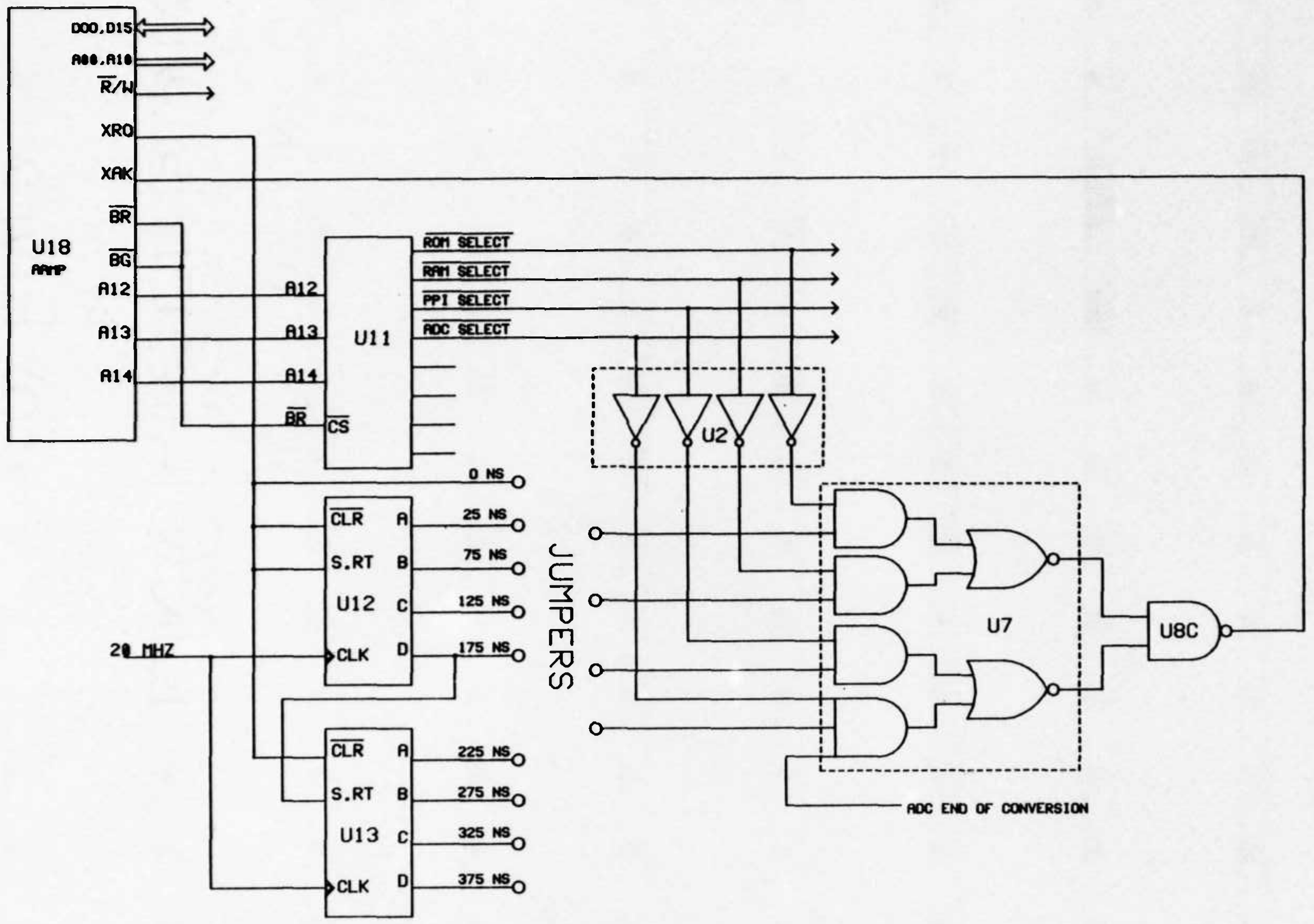


Figure 8. Device Select and Transfer Acknowledge Circuit

Table 6

System Address Assignments

| Address | Purpose |
|---|---|
| ROM | |
| 0000-0008 | Executive Entry Table |
| 0009-000F | not used |
| 0010-0028 | Initialization Procedure |
| 0029-002F | not used |
| 0030-0036 | Trap Procedure |
| 0037-07FF | not used |
| 0800-0FFF | undefined |
| RAM | |
| 1000 | TRAP Number |
| 1001 | User SPCR (where TRAP occurred) |
| 1002 | * User Exception Number |
| 1003 | * User SPCR (where exception occurred) |
| 1004-100F | not used |
| 1010-1019 | User PSD Table |
| 101A-1700 | User Programs and Stack |
| 1701-17F0 | Executive Stack |
| 17F1-17FF | not used |
| 1800-1FFF | not defined |
| PPI | |
| 2000 | Input Buffer |
| 2001 | Output Buffer |
| 2002 | Status Register |
| 2003 | Control Register |
| 2004-2FFF | not defined |
| ADC | |
| 3000 | ADC Start Conversion and Digital Output Buffer |
| 3001-FFFF | not defined |

NOTE: Data, Code, and Local Environment Pointers = 0

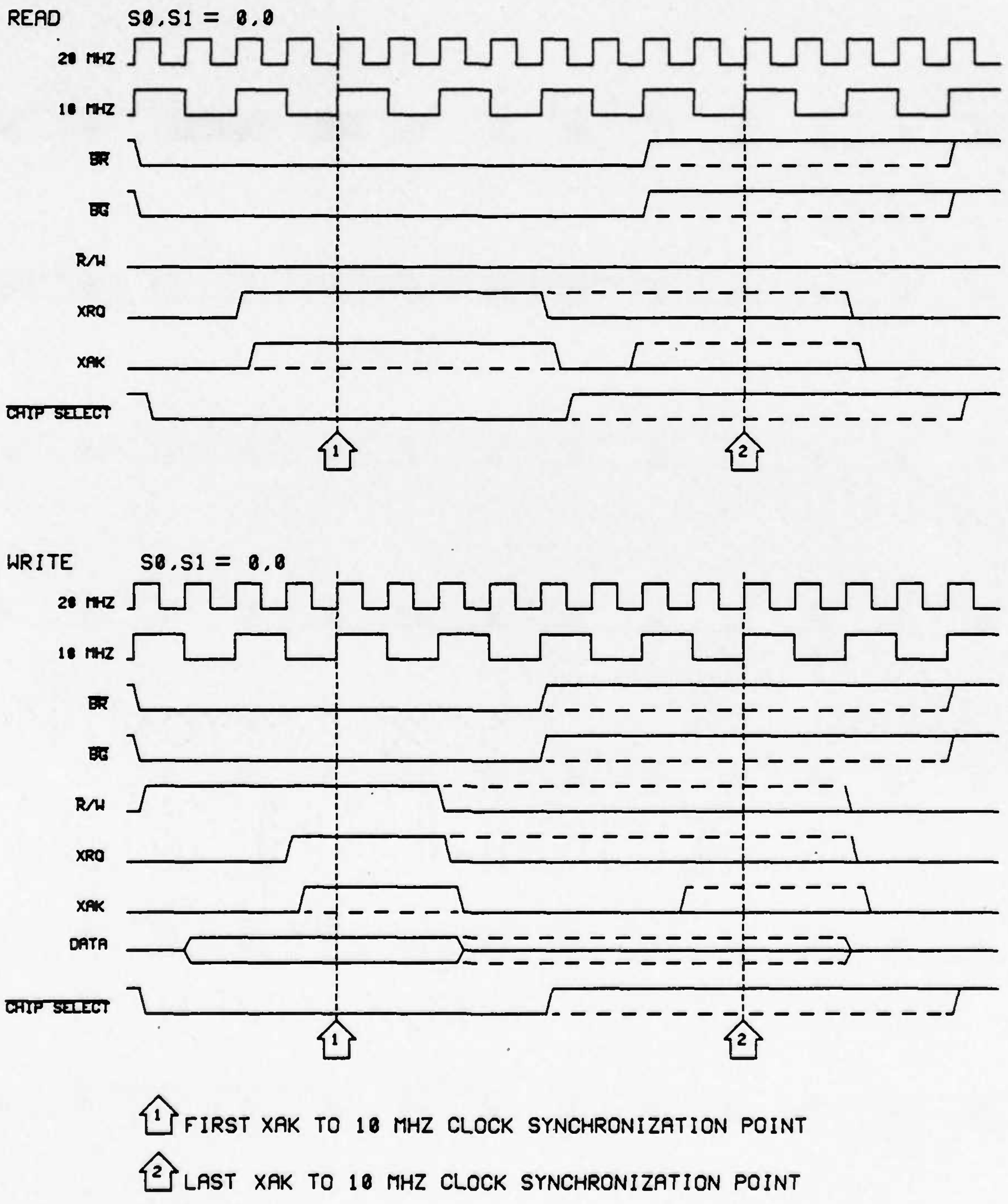
* (ONLY with user exception routine described in Section 3.3)

U11 to provide eight select lines, but only four select lines are used. Table 6 gives the system's address assignments.

A selectable transfer request (XRQ) to transfer acknowledge (XAK) delay circuit is used to generate the XAK signal at a time appropriate to the speed of the device accessed (Figure 8). The circuit allows XAK to be delayed from 15 to 390 ns after a transfer request. Figure 9 shows the possible range of access times with the AAMP configured for the minimum address and data setup time before XRQ, that is S0 and S1 both connected to ground.

When the delay between XRQ and XAK is set to the minimum (15 ns), the system functions the same as if the delay were zero. The delay between XRQ and XAK can range from 0 to 100 ns (minus the propagation delays in the AAMP) for a read operation and from 0 to 50 ns (minus the propagation delays in the AAMP) for a write operation and still synchronize with the 10 MHz clock at the same point. The synchronization points mark #1 in Figure 9 indicate this situation.

The amount of XAK delay chosen depends on the characteristics of the device being accessed. According to the available timing information for the CMOS/SOS AAMP, input data to the AAMP has to be valid at least (4T-150) ns or 50 ns after assertion of XAK. Figure 10 shows that this need not be the case. From any given rising edge of the 10 MHz clock, the assertion of XAK can range from 0 to 100 ns (less propagation delays inside the AAMP) and still synchronize with the next rising edge of the 10 MHz clock. If the assertion of XAK is 95 ns prior to the synchronization point, data would have to be valid 45 ns prior to this point (Figure 10: Case 1); however, if the assertion of XAK is 10 ns prior to the synchronization point, data does not have to be valid until 40 ns after this point (Figure 10: Case 2). The transfer cycle time is the same for both case; therefore, the time when valid data is required at the



READ
S0,S1 = 0,0
20 MHZ
10 MHZ
$\overline{BR}$
$\overline{BG}$
R/W
XRQ
XAK
$\overline{CHIP\ SELECT}$
1
2
WRITE
S0,S1 = 0,0
20 MHZ
10 MHZ
$\overline{BR}$
$\overline{BG}$
R/W
XRQ
XAK
DATA
$\overline{CHIP\ SELECT}$
1
2
1 FIRST XAK TO 10 MHZ CLOCK SYNCHRONIZATION POINT
2 LAST XAK TO 10 MHZ CLOCK SYNCHRONIZATION POINT

Figure 9. Range of Possible Device Access Times

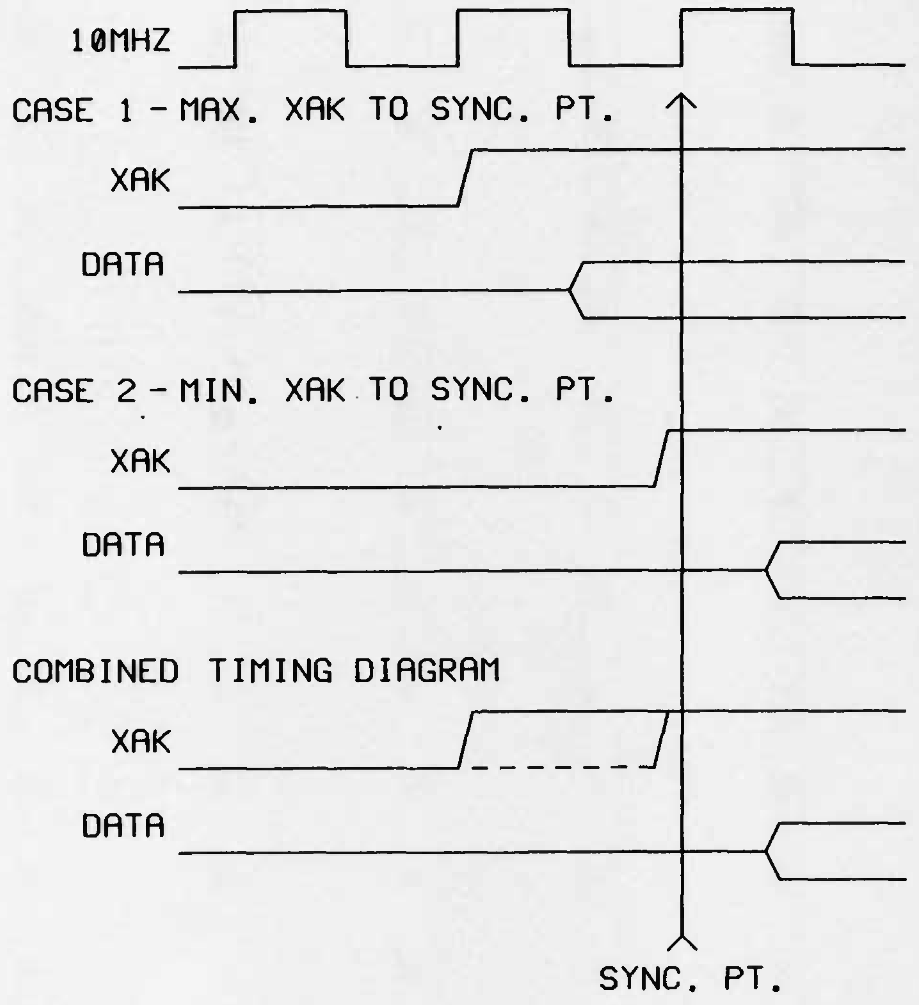

10MHZ
CASE 1 - MAX. XAK TO SYNC. PT.
XAK
DATA
CASE 2 - MIN. XAK TO SYNC. PT.
XAK
DATA
COMBINED TIMING DIAGRAM
XAK
DATA
SYNC. PT.

Figure 10. **Allowable Skew Between Data Valid and Transfer Acknowledge**

Table 7

System Improvement Using Modified Valid Data to AAMP Requirement

| Device | XAK Delay Reduction | Access Time Improvement |
|---|---|---|
| ROM | 0 | none |
| RAM | 25 ns | none (read) |
| | | 100 ns (write) |
| PPI | 50 ns | 100 ns (read) |
| | | none (write) |
| ADC | 0 | none |

AAMP can be based on the synchronization point of XAK to the 10 MHz clock and not just the assertion point of XAK.[7] The developed microcomputer system timing is based on the premise that data be valid no later than 40 to 50 ns after the described synchronization point. In doing so, the system performance is improved (Table 7).

The timing characteristics of the read/write line ($\overline{R}$/W) caused some problems with the initial system design. On a write cycle, the $\overline{R}$/W line goes high 5 to 10 ns after $\overline{BR}$ and $\overline{BG}$ go low and returns low 5 to 10 ns before XRQ goes low. XRQ and $\overline{R}$/W are gated together to generate a write pulse to the RAMs and PPIs; consequently, the removal of $\overline{R}$/W before XRQ caused a 5 to 10 ns read pulse to occur at the end of the write cycle. The glitch on the read enable input caused the PPIs not to program correctly but had no apparent effect on the RAMs. Available AAMP specifications indicate that $\overline{R}$/W and XRQ are removed simultaneously. Internal chip delays in the AAMP probably account for the skewing effect. Since the $\overline{R}$/W timing characteristic was discovered after the microcomputer system was designed and built, the simplest way to remove the glitch problem was to delay $\overline{R}$/W by passing it through inverters connected in cascade. Inverters U2B and U3B were added to the design. Thus $\overline{R}$/W passes through redundant inverters U1E and U3B before it is used by the RAM and PPI select circuits (Figure 7).

3.2.3 Microprocessor to Read-Only Memory Interface -- The read-only memory (ROM) is required to store the Executive Entry Table and programs required by the microcomputer system at power turn-on. The Executive Entry Table has to reside at addresses "000000" through "000008". Procedure pointers in the Executive Entry Table define where the executive programs lie in memory.

Two CMOS 2048x8-bit UV erasable and electrical reprogrammable read-only memories (EPROMs) were used to realize a 2048-word ROM. National Semiconductor NMC27C16Q-35 EPROMs were used which have a 350 ns access time. The microprocessor to ROM interface is shown in Figure 11. The timing diagram is given in Figure 12.

The ROM is selected when address lines A14, A13, and A12 are "000" and bus request ($\overline{BR}$) is low. The output from the ROM is enabled only when $\overline{R}$/W is low, XRQ is high ,and the ROM select is low. The design allows for processor write attempts to the ROM addresses without causing damage to the system. Write attempts generate a XAK but do not enable the ROM output, that is, the output remains in a high-impedance state.

The delay between XRQ going high and XAK going high is 240 ns. The amount of delay chosen depends on when the data from

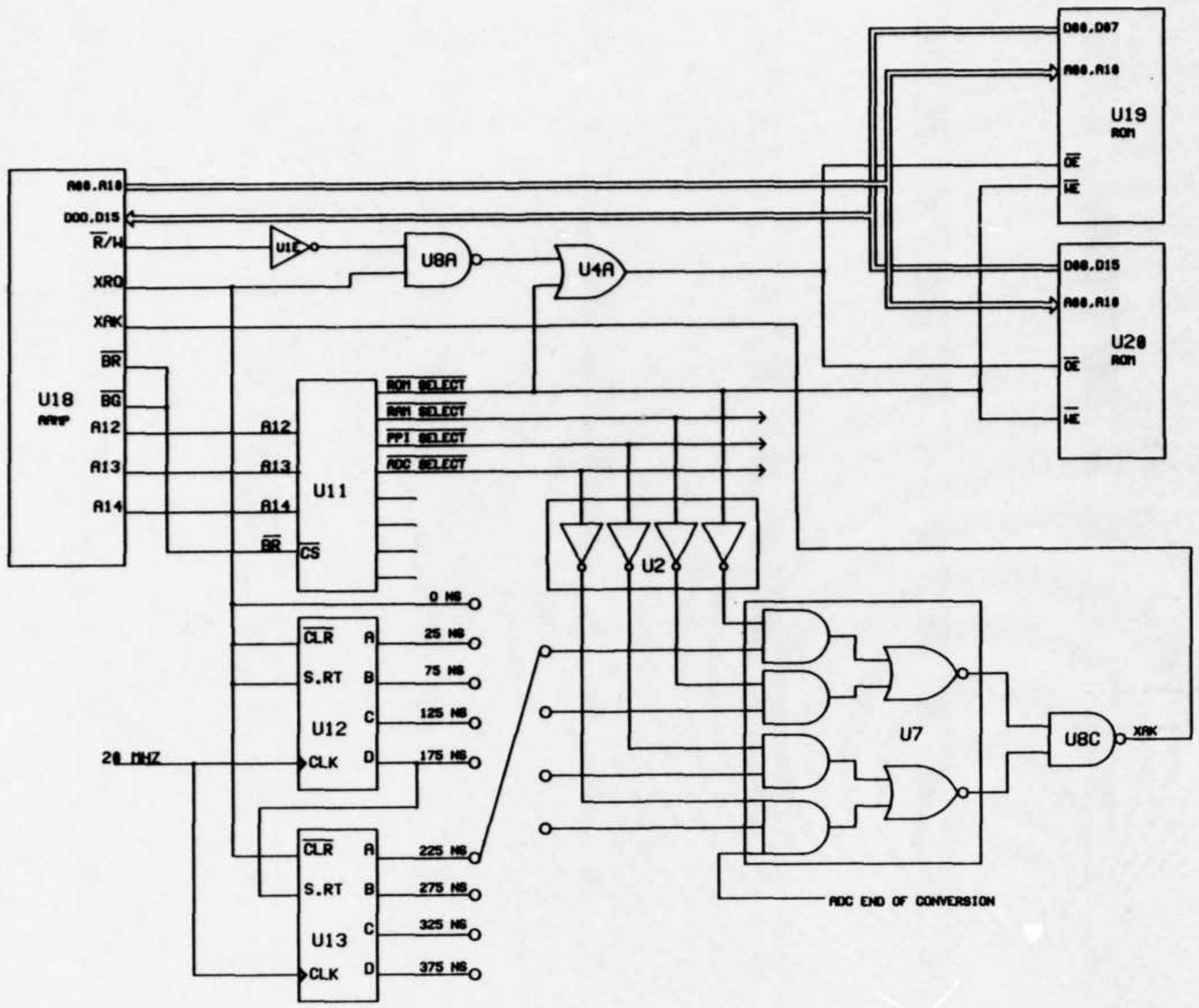


Figure 11. Microprocessor to Read-Only Memory Interface

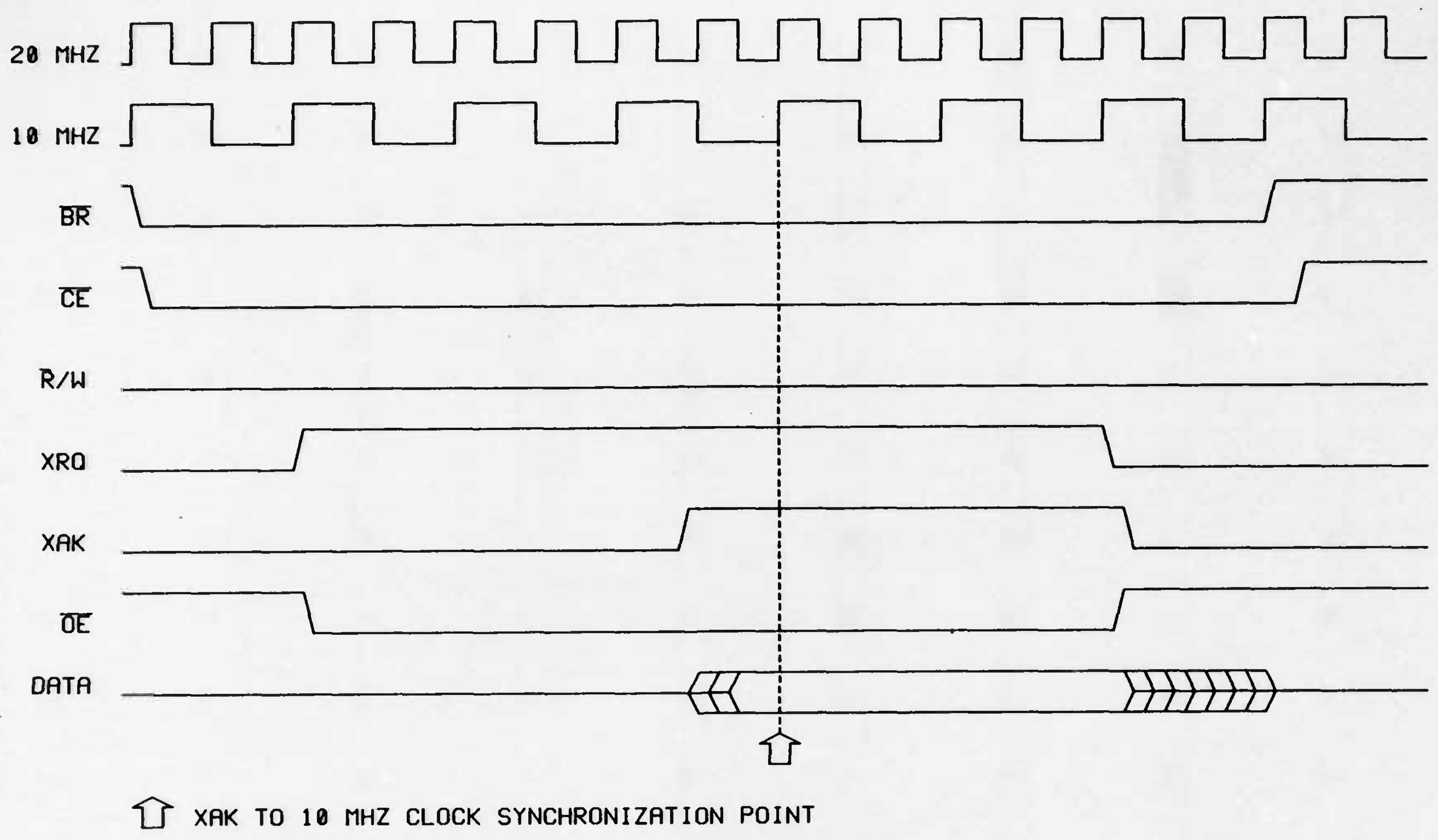


Figure 12. Microprocessor to Read-Only Memory Timing Diagram

the ROM becomes valid. The first possible synchronization point of XAK to the 10-MHz clock is shown in Figure 12.

3.2.4 <u>Microprocessor to Random Access Memory Interface</u> -- The random access memory (RAM) is required for executive stack and data storage. The microcomputer system also uses RAM to store user programs, stacks, and User PSD Tables.

Two Hitachi HM6116P-2, 120 ns access time, static CMOS 2048x8-bit RAMs were used to form a 2048-word random access memory. The AAMP to RAM interfacing is shown in Figure 13 and the timing diagrams are given in Figure 14.

A "001" on address lines A14, A13, and A12 provides a RAM select signal from decoder U11. The RAM select, $\overline{R}/W$, XRQ, and a delay of XRQ are used to generate a RAM output enable and a RAM write enable. When the RAM select is low and XRQ is high, the output of NOR gate U10A enables signals to pass through NAND gates U8B and U8C. The output of U8B is low only when the output of U10A is high and $\overline{R}/W$ is low. The output of U8C is low only when the output of U10A and $\overline{R}/W$ are high. Thus, the RAM output enable and write enable can not be active at the same time. On a write operation, the write enable ($\overline{WE}$) must be removed at least 5 ns prior to data invalid. The write enable ($\overline{WE}$) and a 125 ns delay of XRQ is combined through OR gate U4B to force $\overline{WE}$ high approximately 20 ns before the data goes invalid. The length of the resultant write pulse is approximately 105 ns which also satisfies the required minimum length of 70 ns.

The $\overline{R}/W$ line is normally in the low or read state. The $\overline{R}/W$ line returns from the write to read state 5 to 10 ns before XRQ goes low. If the two redundant inverters U1E and U3B are removed

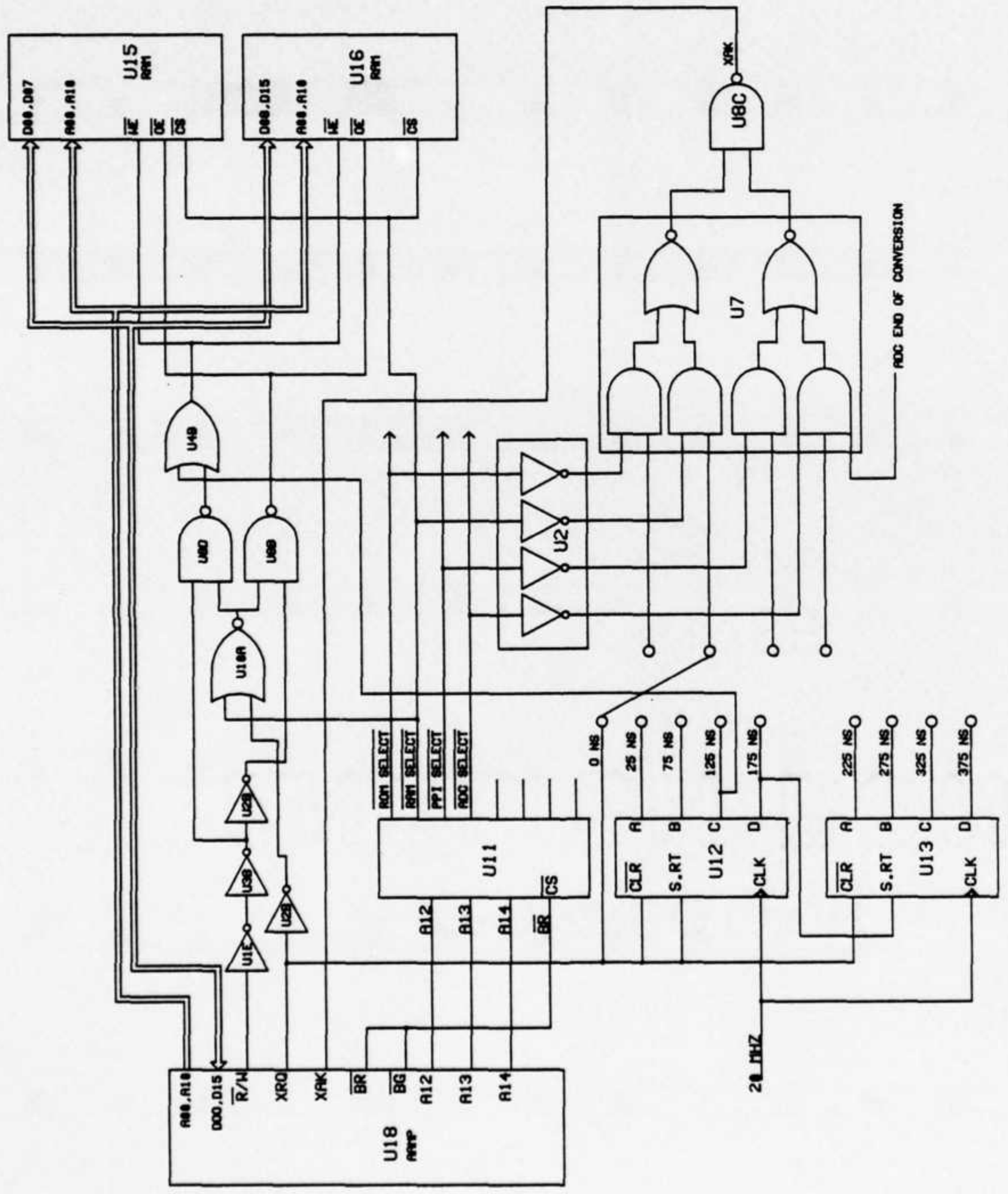
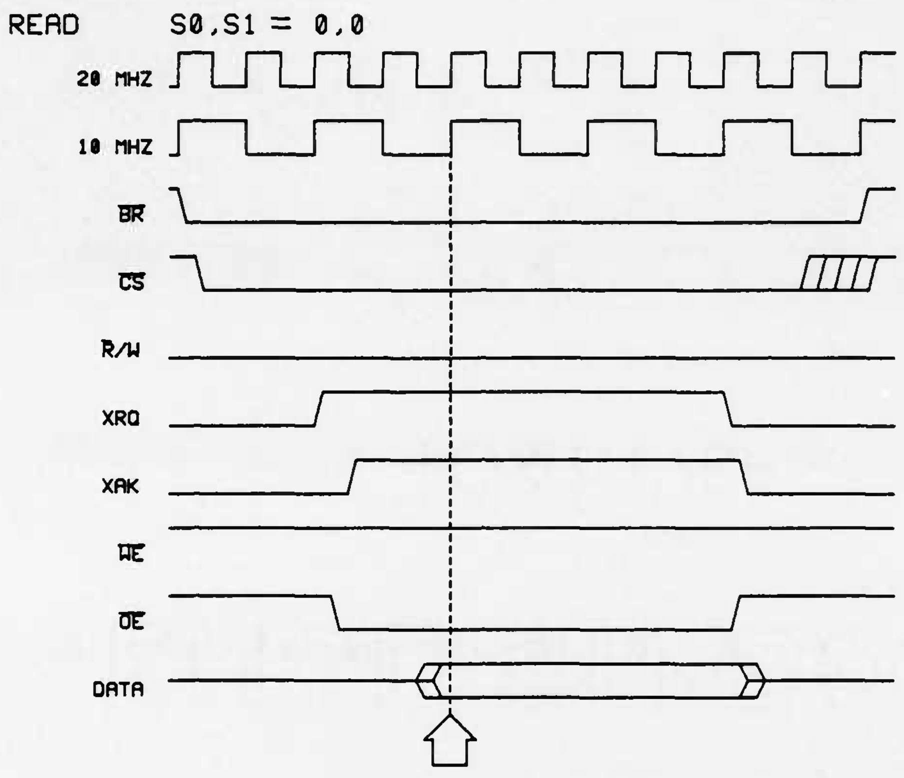


Figure 13. Microprocessor to Random Access Memory Interface



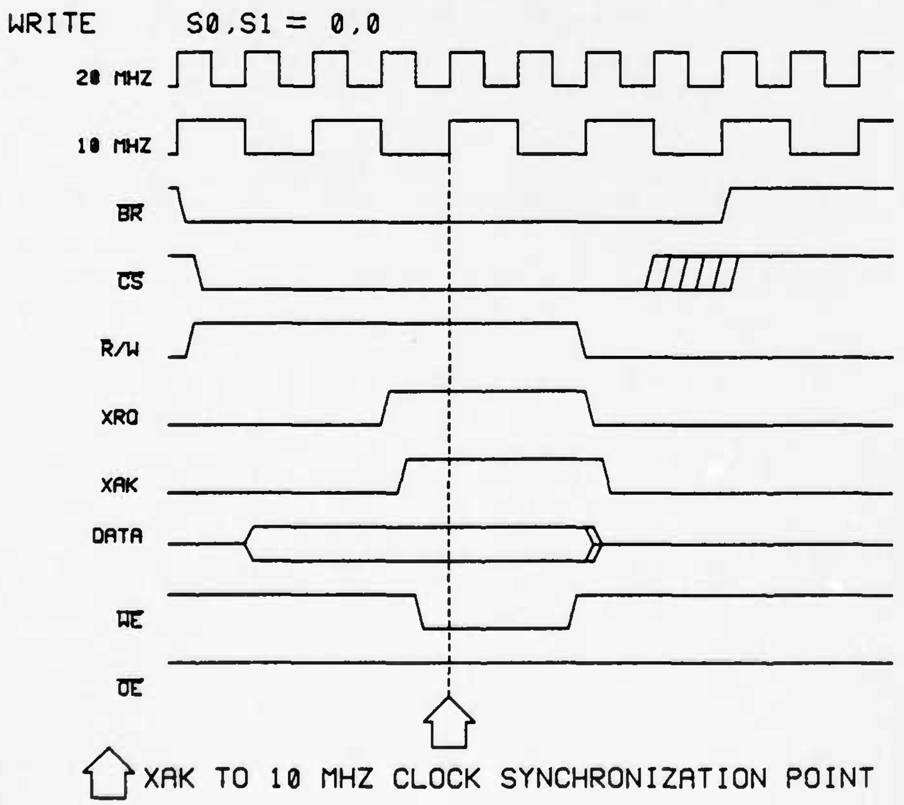


Figure 14. Microprocessor to Random Access Memory Timing Diagrams

from the circuit, the output of U10A would still be a logic high when the change in the read/write state is felt at the inputs of U8B and U8C. The change does not influence write enable ($\overline{WE}$) because it has already been disabled by the XRQ delay into U4B; however, the change causes a small glitch on the RAM output enable line ($\overline{OE}$). The propagation of $\tilde{R}$/W through the added inverters insures that the output of U10A is low before the change in the read/write state is felt at U8B and U8C.

The delay between XRQ and XAK is determined by finding the allowable delays possible for the read cycle and observing the effect they have on the write cycle timing. On the READ timing diagram in Figure 14, the first possible synchronization point of XAK with the 10 MHz clock is marked with an arrow. A delay of 15 or 40 ns ( values include 15 ns for the propagation delay through U7 and U8) will both be synchronized at the indicated point. From the WRITE timing diagram in Figure 14, note that the use of a 40 ns delay will cause the indicated synchronization point to be missed, thus a delay of 15 ns is the optimal choice. Missing the synchronization point would cause a 100 ns increase in the write cycle time.

The use of the 15 ns delay between XRQ and XAK allows for earlier synchronization but it violates the manufacturer's suggested allowable skew time between the assertion of XAK and data valid on a read cycle. The skew time would not be violated if a 40 ns delay were used. Since both delays result in the same read cycle time, the violation does not influence the AAMP's operation as was shown in Section 3.2.2.

3.2.5 *Microprocessor to Parallel Port Interface* -- A peripheral interface is required to facilitate communication to external systems. A programmable parallel interface was chosen for this purpose. Two 8-bit, Harris 82C55A, programmable peripheral interfaces (PPIs), are used to form a 16-bit parallel interface. Each PPI has 24 programmable input/output lines which can be used in three major modes of operation: mode 0, basic input/output; mode 1, strobed input/output; and mode 2, bidirectional bus input/output. The various modes are selected by writing to a control register.

The programmable peripheral interfaces are connected in the microcomputer system as shown in Figure 15. Each PPI is programmed for operation in mode 1 and provides an 8-bit output, an 8-bit input, four handshaking control lines, two microprocessor interrupt lines, and two general purpose input/output lines. Together, the two PPIs provide 16 lines of parallel output, 16 lines of parallel input, four handshaking lines, two microprocessor interrupt lines, and four general input/output lines. Four control lines and two interrupts are redundant and are not considered for use. The input data buffer address is "002000"; the output data buffer address is "002001"; the PPI status register address is "002002"; and the control register address is "002003". Appendix C explains how to program and configure the PPIs for operation in mode 1. A control word of "BCBC" configures the PPIs as described with the general input/output lines configured for the input direction.

A "010" on address lines A14, A13, and A12 decodes to select the PPIs. The write and read enable circuit is the same as that

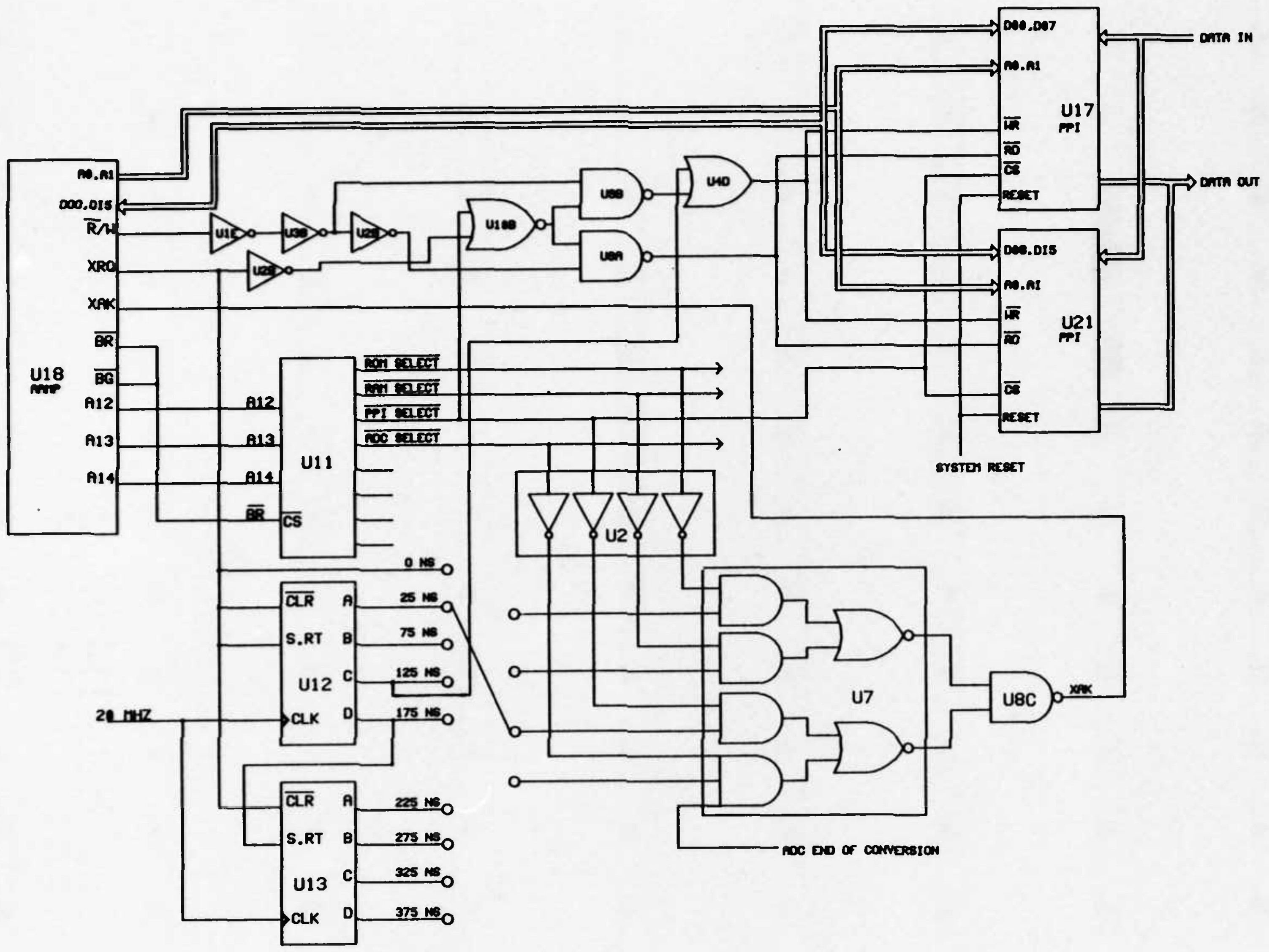


Figure 15. Microprocessor to Parallel Port Interface

for the RAMs. See Section 3.2.4 for a description of operation. The read glitch on the end of a write cycle described in connection with the RAM select circuit also applies to the PPI select circuit. In fact, the PPIs would not program (configure) correctly until the glitch was removed.

To determine the optimum XRQ to XAK delay, the minimum delay times were first found for the read operation. The final choice was determined by the write operation timing restraints. From the read timing diagram in Figure 16, data becomes valid approximately 20 ns after the marked synchronization point; therefore, the synchronization of XAK to the 10 MHz clock can only occur at the indicated point or later. A 90 ns XRQ to XAK delay results in the indicated synchronization point being missed because of propagation delays inside the AAMP. Thus XRQ to XAK delays of 15 and 40 ns were both considered for use. Note that both violate the XAK to data skew time restriction (Table 5), but the use of either will not affect the read cycle time.

The PPIs require a 100 ns minimum write pulse length and a 60 ns minimum data/address hold time following the removal of the write pulse. A 15 ns XRQ to XAK delay and a 125 ns XRQ delay to U4D will generate a write pulse length of approximately 105 ns and a data/address hold time of only about 20 ns. A 40 ns XRQ to XAK delay and a 125 ns XRQ delay to U4D will provide a write pulse length of 105 ns and a data/address hold time of 100 to 110 ns. Thus, the later is used.

The write timing diagram in Figure 16 shows XAK to 10 MHz clock synchronization points for both 15 ns and 40 ns XRQ to XAK

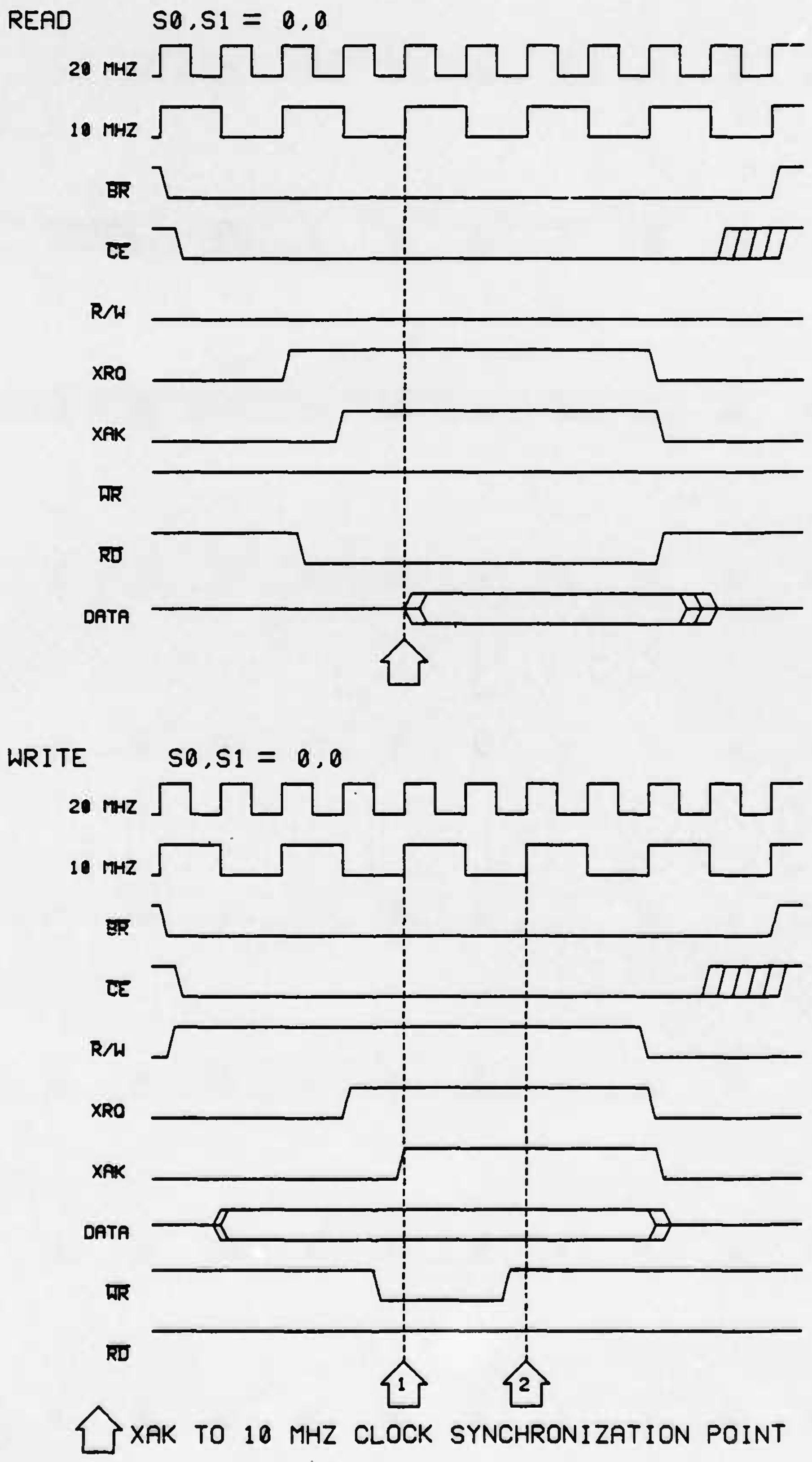


**Figure 16. Microprocessor to Parallel Port Timing Diagrams**

delays which are marked #1 and #2 respectively. Note that the 40 ns XRQ to XAK delay results in XAK going high too close to synchronization point #1 which results in a 100 ns delay and synchronization at point #2.

3.2.6 <u>Microprocessor to Analog-to-Digital Interface</u> -- The analog -to-digital conversion section of the microcomputer system provides a means of sampling real-time data for digital processing. An analog-to-digital converter (ADC) is used with a sample-and-hold to do the conversions (Figure 17).

An Analog Devices AD583 sample-and-hold and a Datel Intersil AD-HC12B 12-bit analog-to-digital converter are used. The sample-and-hold and analog-to-digital converter operate from -15 and +15 V power sources; therefore, digital logic level shifters are required for interfacing these devices into the system. The 12 digital outputs, the end-of-conversion ($\overline{EOC}$), and the start conversion signal are all passed through level shifters. The HOLD signal to the sample-and-hold is designed to operate with 0 to 5 V input levels and does not require level shifting. The ADC is configured to provide a two's complement digital value at its output. $\overline{EOC}$ clocks the digital output into the three-state D-type flip-flops (U25 and U26) which form an addressable data register to the AAMP.

A "011" on address lines A14, A13, and A12 will select the analog-to-digital section of the microcomputer system. If the prior conversion is complete ($\overline{EOC}$=0), the three-state flip-flop register can be accessed like any memory location. The data from the register becomes valid on the bus approximately 20 ns after

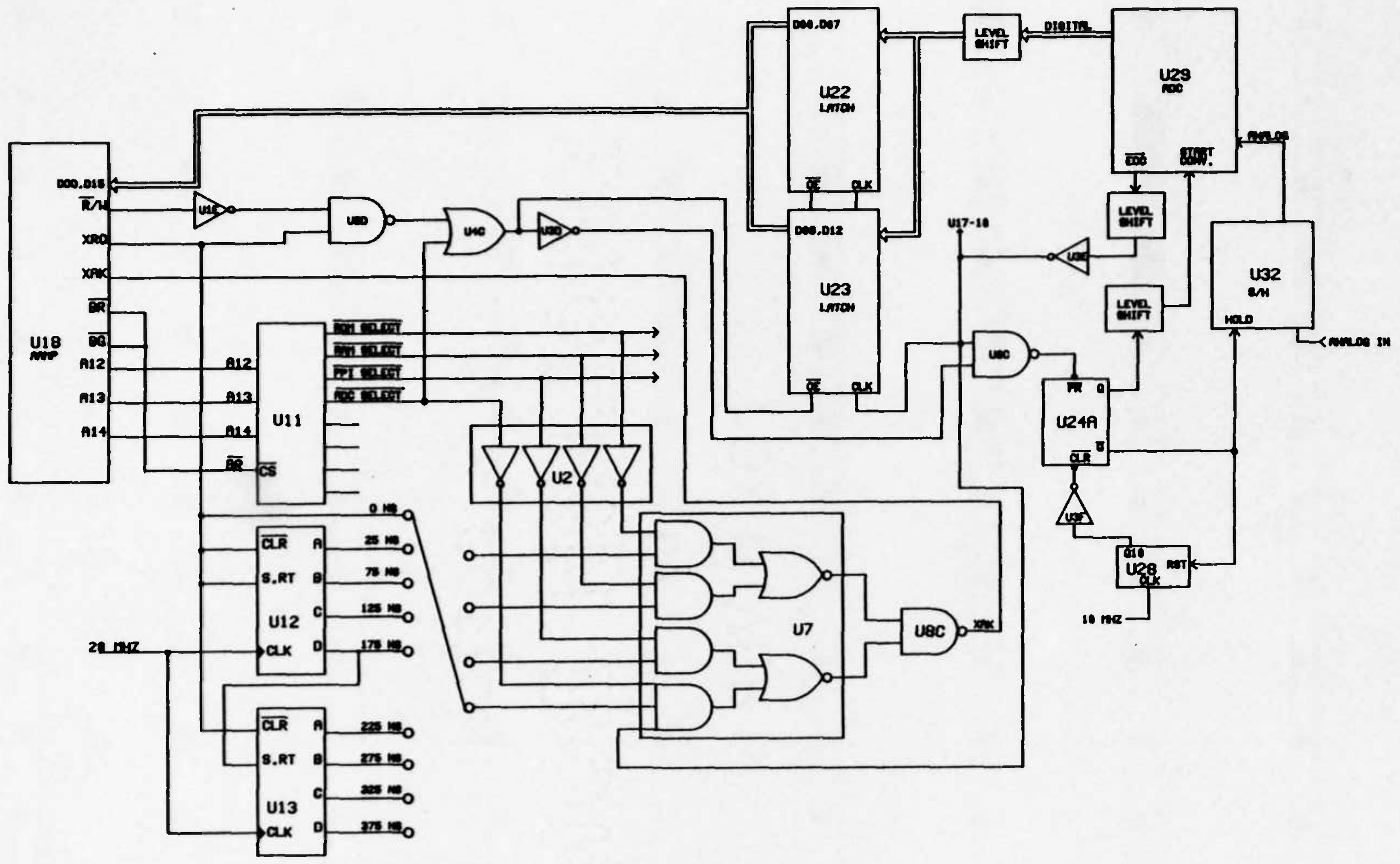


Figure 17. Microprocessor to Analog-to-Digital Section Interface

the output enable goes low; therefore, no XRQ to XAK delay is required. See Figure 18 for the timing diagram.

The ADC is configured so that it goes into a low-power standby mode when no conversion is taking place. When the addressable data registers U22 and U23 are accessed by the processor, the D-type flip-flop U24A is set by the output of NAND gate U9C. The Q output of U24A is the start conversion signal to the ADC. The required start conversion pulse width is obtained by using counter (U28) to reset the flip-flop. The $\bar{Q}$ output of U24A is used to control the sample-and-hold (U30). The HOLD signal to the sample-and-hold is the complement of the ADC start conversion signal, thus the sample-and-hold only samples data during the time the start conversion signal is high and is in the hold mode the rest of the time.

The analog-to-digital conversion section of the microcomputer system is selected anytime address lines A14, A13, and A12 are "011". If a write operation occurs, NAND gate U8D prevents the analog-to-digital conversion section from responding to the ADC select from U11; however, the XAK signal is generated to prevent a system failure.

The first read operation to the analog-to-digital section starts the first conversion cycle and subsequent read operations return valid data. The system was designed so that the AAMP could consecutively access the analog-to-digital conversion section; however, the system fails to operate properly. The EOC line is connected to U7 in the XAK generation circuit to delay the generation of XAK until the ADC is finished with the current

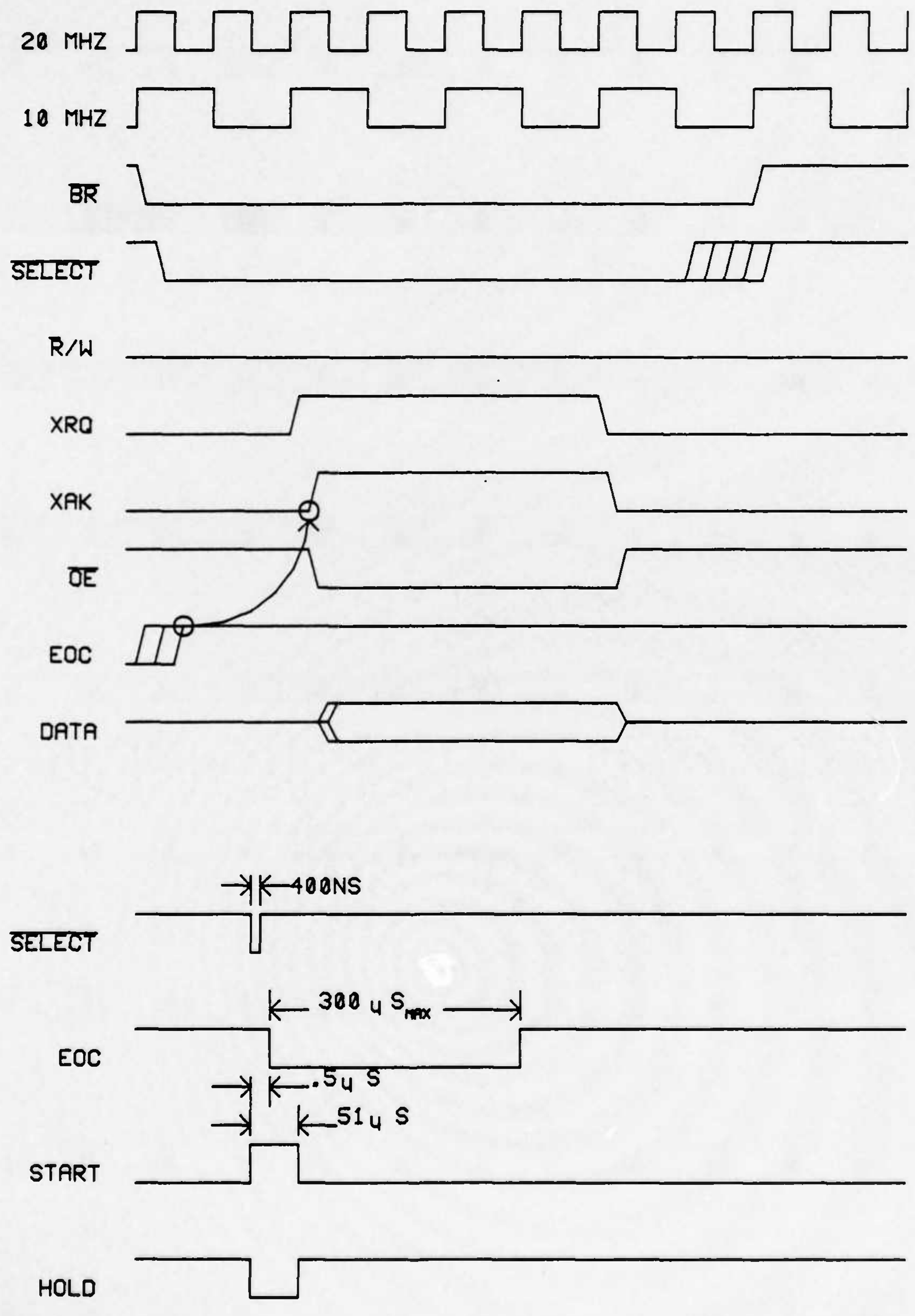


Figure 18. Microprocessor to Analog-to-Digital Section Timing Diagram.

conversion and is also connected to the data register U22 and U23 to latch the ADC output when the conversion is complete.

Two problems were encountered when the analog-to-digital section was accessed repeatedly without any delay between accesses. First, the AAMP is fast enough that the EOC line does not have time to go low before the second read occurs. The circuit will initiate another conversion cycle and the processor will receive invalid data. There must be approximately 500 ns between reads to the ADC to prevent this problem, but a second problem then occurs. If the subsequent read operation occurs after EOC goes low but before the conversion is finished, the conversion will terminate correctly, the processor receives valid data, and a new start pulse is generated as designed. The problem is that the ADC does not respond to the start pulse and the end-of-conversion signal remains high. The start pulse occurs approximately 20 ns after the EOC goes high, but the ADC fails to respond. The start pulse must occur too soon following the previous conversion.

To remedy the problems, the EOC line has been connected to U17-10, a general purpose input/output line on the PPI U17. The PPIs are configured so that the general purpose input/output lines are in the input mode. Thus the processor can poll the PPI status register to determine the state of the EOC line. Bit 7 of the PPI status word corresponds to the state of the EOC line. A "zero" indicates EOC is low (conversion in progress), and a "one" indicates EOC is high (conversion finished). If the time between accesses to the analog-to-digital section is less than approximately 0.35 ms, software polling of the EOC state must be added to the data acquisition program. The suggested method is to poll for an EOC transition from a "high" to "low" before continuing normal program execution and to poll for an EOC transition from a "low" to "high" before accessing the analog-to-digital section the next time.

An alternative method to correcting the problem would be to change the hardware design. This approach would increase the chip count and circuit complexity and would not appreciably change the maximum possible conversions per second. Thus, correction by hardware is not viable solution.

3.2.7 Microcomputer to Host Computer Interface -- The Hewlett-Packard System 45 Desktop Computer (HP-9845B) was used as the host computer to the AAMP based microcomputer system. The computer interface chosen was the HP-98032A general purpose, 16-bit input/output interface (GPIO). A 50-pin edge connector on the microcomputer board serves as the connect point between the two systems. The communication process is accomplished using a handshake protocol. Multiplexers (U14-A/B/C) are used to connect the GPIO peripheral control lines to the PPI input and output handshake lines (Figure 19). All outputs from the GPIO are open-collector; therefore, resistive line terminators U32 and U33 are required as shown in Figure 7.

The GPIO has three control lines ($I/\overline{O}$, PCTL, and PFLG), two input lines (STI 0 and STI 1), two output lines (CTL 0 and CTL 1), 16 data input lines, and 16 data output lines. $I/\overline{O}$ indicates the direction of the transfer with respect to the HP system and is used to control the multiplexers. The logic level of PCTL and

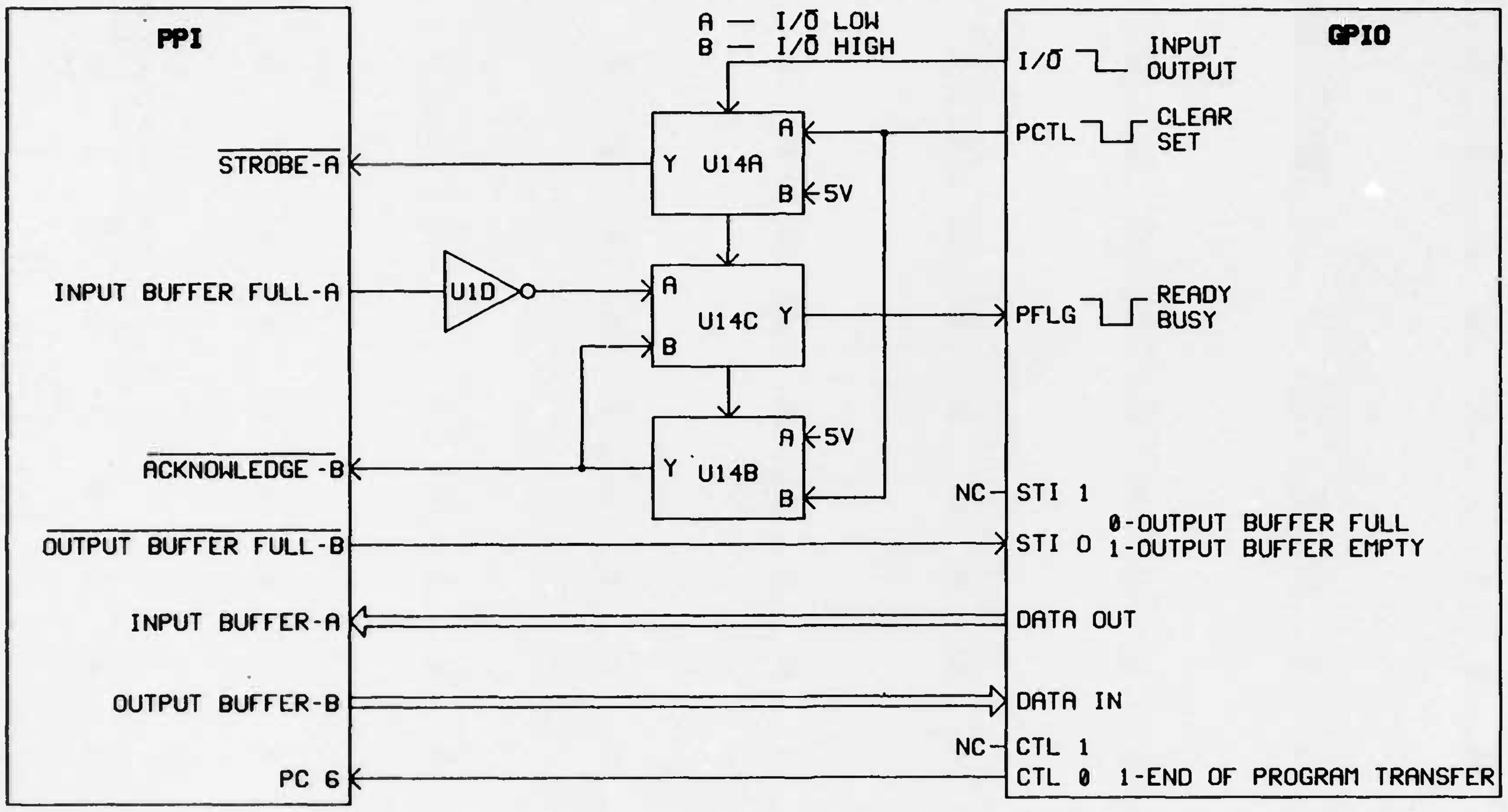


Figure 19. Microprocessor to Host Computer Interface

PFLG can be complemented by installing jumpers in the GPIO. PCTL was configured such that a "high" means CLEAR and a "low" means SET. PFLG was configured where a "high" means the microcomputer system is READY and a "low" means the microcomputer system is BUSY. The PPI output buffer full line is connected to the GPIO input STI 0 with a "high" meaning the buffer is EMPTY and a "low" meaning the buffer is FULL. GPIO output line CTL 0 is connected to PPI input line PC6 (U17-11). This line is used to indicate to the microcomputer system when the program transfer from the host computer is complete (see Section 3.3).

When the microcomputer system to GPIO interface was designed, a choice had to be made on how to handle transfers of data from the microcomputer system to the host computer. If the PPI output buffer full line is connected to the PFLG line, interrupt handling is required in the HP computer in order to know when to read data from the microcomputer system. A preferred alternative was to use software polling of the PPI output buffer full line. The later approach simplifies the programming required in the HP computer and is used. To implement this approach, the PPI output buffer full line was connected directly to the GPIO input STI 0.

The $I/\overline{O}$ line is normally in the INPUT state which means PCTL is connected to both the PPI output buffer acknowledge line and PFLG through U14B and U14C. When PCTL is CLEAR, PFLG gets an indication that the microcomputer system is READY for the next transfer. When PCTL is SET, PFLG gets an indication that the microcomputer system is BUSY or not ready for the next transfer. The configuration completes the required handshake to the GPIO but does not actually indicate the state of the PPI output buffer. The host computer has to check the state of input line STI 0 to determine when the PPI's output buffer is full.

To write data to the PPI, the GPIO puts a "low" on $I/\overline{O}$ and data on the output bus before setting PCTL. The "low" on $I/\overline{O}$ connects PCTL to the PPI input buffer strobe line in U14A and connects the PPI input buffer full line to PFLG in U14C. When PCTL goes SET, the data is clocked into the PPI; and the PPI input buffer full line goes "high" which corresponds to a BUSY on PFLG. The host computer is then free to do an input or wait for the microcomputer system to clear the PPI input buffer.

The GPIO has two modes of operation: full and pulse. Either mode can be used. The reader is referred to the HP-98032A Installation and Service Manual for further information on the GPIO.[8] The GPIO jumpers that are required for proper operation are listed in Appendix D along with the pin assignment for the microcomputer system edge-connector.

## 3.3 <u>System Software Description</u>

The software, for controlling the microcomputer system on power turn-on and system reset, resides in ROM. The software allows programs for execution to be transferred from the host computer (HP-9845B). The transferred programs are stored in RAM and executed in the user mode.

Two initialization procedures were developed and tested. One uses maskable interrupts and the other uses software polling of the PPI status register to determine when the host computer has transferred a word into the PPI. The interrupt method was implemented only to identify any problems associated with the use of interrupts. The adopted procedure for the system uses software polling of the PPI status register. Appendix E gives a listing of the later initialization procedure.

The initialization procedure is written such that the user program must be transferred from the host computer in a code/address sequence. A single output line from the host computer tells the microcomputer system when the transfer is complete. The output line is connected to PPI U17-11 (input line PC6). Bit 6 of the PPI status register is a logic "1" when the program transfer is in progress and is a logic "0" when the program transfer is complete. When the initialization procedure detects a zero on the end-of-program transfer line, the initialization procedure stores a User PSD Pointer value "1010" in the initial executive top-of-stack location (TOS) and does a outer-procedure return which causes the processor to context switch to the user mode and execute the transferred program. The system software is not set-up to allow multitask scheduling, but a user program can have multiple subprograms. The User PSD Table must be transferred to locations "001010" through "001019". The user program and stack locations must lie within the space specified in the memory map (Table 6).

The AAMP determines which executive procedures are defined from the Executive Entry Table. If a procedure ID (PROCID) is zero, the processor will halt when that procedure is called. The only procedures presently defined in the microcomputer system are the initialization and trap procedures.

A user program's execution can be halted by the execution of a trap instruction or by the processor if a fatal error occurs. When the processor aborts a user program, it records the state of the processor in the User PSD Table and passes a trap number to the executive trap handler. If no trap handling procedure exists, the processor will halt and must be reset.

To aid in debugging user programs, a method was developed which tells the host computer where the user program's execution stopped and what the associated trap number was. A system reset will destroy this information, so a trap handling procedure is used to store the trap number and the user syllable program counter (SPCR) in RAM addresses "001000" and "001001", respectively. The trap handler saves the described information and then halts the processor by executing the HALT instruction. The microcomputer system can then be restarted, and a user program can be transferred from the host computer to retrieve this information.

When the execution of the user program results in an accumulator overflow or underflow, the processor looks to the User PSD Table for the exception handler PROCID. If no exception handler exits, the result for the mathematical function which caused the exception will be as described in the instruction manual.[3] If an exception handler is defined, the processor passes an exception number to the exception handler. The exception number identifies the type of operation which caused the exception. Exceptions are generally undesirable and indicate a programming problem; therefore, an exception handler was developed to store the user exception number and user SPCR in

RAM addresses "001002" and "001003", respectively. The exception handler then passes a trap number value of eight to the executive trap handler. The executive trap handler halts the processor as described above. Appendix F gives a listing of the exception handler. The user program, which retrieves the trap number and associate user SPCR, can also be used to retrieve the exception number and associated User SPCR. Appendix F also gives a listing of such a program.

The microcomputer system has two switches labeled "start" and "stop". The start switch is gated to the processors reset input and will always cause a cold restart because the system Continuation Status Word is zero. The stop switch is connected to the nonmaskable interrupt by a user installed jumper. The nonmaskable interrupt PROCID is zero which means no procedure is defined. When the stop switch is activated, the processor aborts what it is doing and halts. Thus the two switches function as "run" and "stop" switches.

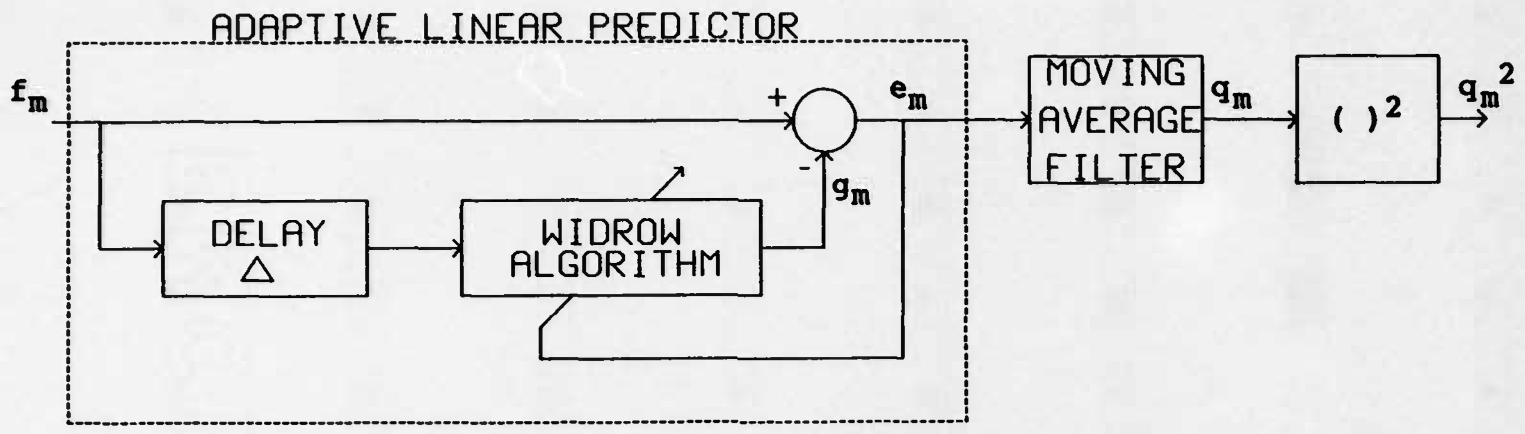
If a user program's execution is completed normally (by way of an outer-procedure return from the user mode), a trap code "0" will be passed to the trap handler and the user SPCR will be reset to zero. The trap handler will store the trap number and SPCR as described above and halt the processor. If the user program's execution is halted by the stop switch, the state of the user program is recorded in the User PSD Table and a context switch to the executive mode is made. Since the nonmaskable PROCID is zero, the processor halts. In both cases, the user program can be re-executed or restarted without the need of transferring it again from the host computer. First, the microcomputer system has to be reset (restarted). The initialization procedures must receive one code/address pair before it checks the end-of-program transfer line; therefore, the host computer must set the end-of-program transfer line low and send a "dummy" code/address pair. Any unused RAM address or any ROM address can be used for the "dummy" address. The initialization procedure then stores the address offset to the User PSD Table in the User PSD Pointer location and executes a RETURN instruction which causes the processor to context switch to the user mode. The user program will either start from the beginning or continue from the point of interruption depending on the stored value of the user SPCR.

## 4. Widrow Adaptive Linear Predictor Algorithm Implementation

The simplest method to compare the performance of microprocessors is to implement the same algorithm on each and compare results. The Widrow adaptive linear predictor algorithm was previously implemented on several other microprocessors at Kansas State University and thus was chosen as a benchmark program for the AAMP. The Widrow adaptive linear predictor algorithm involves a large number of multiplications and array handling which is representative of algorithms for many applications in which the AAMP may be used. Figure 20 shows a block diagram of the algorithm.

Kenneth Albin did the ground work for implementing the Widrow adaptive linear predictor algorithm on the AAMP based microcomputer system.[2] He encoded four versions of the algorithm of which three were implemented. They include an Ada to AAMP compiled floating-point version, an Ada to AAMP compiled fixed-point fractional version, and a hand coded fixed-point fractional version. The later eliminates much of the array updating required in the other two versions. Kenneth Albin's work was very comprehensive; but because no AAMP based microcomputer system existed on which the programs could be tested with real-time data; a couple of minor problems went undetected. Appendix G contains a description of these problems and revised listings.

Software for the HP-9845B computer was developed to facilitate the microcomputer system testing. Program file generation, program file editing and storage, data file generation, program and data file transfer, and received data



$$g_m = \sum_{k=1}^{16} b_{m,k} * f_{m-k} \qquad q_m = \frac{1}{16} \sum_{k=1}^{16} e_{m-k+1}$$

$$e_m = f_m - g_m \qquad q_m^2 = q_m * q_m$$

$$b_{m+1,k} = u * b_{m,k} + v * e_m * f_{m-k}$$

**Figure 20. Widrow Adaptive Linear Predictor Algorithm Block Diagram**

file plotting are among the routines developed. These routines have not been considered essential to this thesis and are not discussed.

To verify proper operation of the Widrow linear adaptive filter algorithm, an existing FORTRAN version was translated into BASIC for use in the HP-9845B (host computer). Sample data files were processed on both systems and the outputs were plotted for comparison. The plotted results from the AAMP encoded floating-point version were identical to those from the HP-9845B. The plotted results from the two fixed-point versions had the same general shape as those from the floating-point version, but of course differed in scale. The plotted results from the two fixed-point versions were however identical with each other which confirmed their equivalence. Thus the operation of the three AAMP encoded Widrow linear adaptive algorithms was verified.

# 5. Microcomputer System Performance

The performance evaluation of the AAMP based microcomputer system encompasses four areas: system performance compared to other processors, analog-to-digital sampling rate, peripheral data transfer rate, and power consumption.

## 5.1 AAMP Based Microcomputer System Compared to Other Processors

The Widrow adaptive linear predictor algorithm was used to benchmark the microcomputer system. The maximum execution rate was determined for the three versions of the algorithm described in Appendix G. The programs were stripped of all instructions dealing with the interfacing to the host computer. The input was simulated by reading from the analog-to-digital conversion section. The output was simulated by writing to the PPI. Table 8 shows the comparison of the AAMP's performance to that of other microprocessors. The information in the table comes from the work of Kenneth Albin with the exception that the estimates of the AAMP's performance are replaced by the actual values.[2] The number of iterations given for the AAMP are nearly the maximum possible. With the exception of the storing of the output to the PPI, the AAMP executed the algorithm in essentially a no-wait state configuration because the program was completely executed from RAM. If the programs were executed from ROM, each instruction fetch would require an additional 200 ns. The estimated iteration rates for execution of the algorithm from ROM are also given in Table 8.

Table 8

Widrow Adaptive Linear Predictor Algorithm
Execution Rates

| Processor | Iterations/ second | | Clock rate (MHz) | Multiply # bits | / type | / time(us) |
|---|---|---|---|---|---|---|
| Z80 | 130 | | 4 | 8 | SW,fixed | 147.25 |
| Z80 | 47 | | 4 | 16 | SW,fixed | 554.25 |
| NSC800 | 296 | | 4 | 8 | HW,fixed | |
| NSC800 | 273 | | 4 | 16 | HW,fixed | |
| 8748 | 80 | | 6 | 8 | SW,fixed | 252.5 |
| AAMP (standard) | 770 | (note 1) | 20 | 16 | HW,fixed | 4.75 |
| AAMP (modified) | 817 | (note 2) | 20 | 16 | HW,fixed | 4.75 |
| AAMP (standard) | 345 | (note 3) | 20 | 24 | HW,float. | 19.15 |

(note 1: Execution from RAM -- Execution from ROM would be 757)
(note 2: Execution form RAM -- Execution from ROM would be 804)
(note 3: Execution form RAM -- Execution from ROM would be 341)

## 5.2 Analog-to-Digital Sampling Rate

The maximum sampling rate obtainable from the Datel Intersil Model ADC-HC12B, analog-to-digital converter (ADC), was determined as follows. A known frequency sinewave was applied to the system. A program was executed with the following sequence:

(1) read value from the ADC

(2) poll for the end-of-conversion line "low"

(3) store value from step (1) in an array

(4) poll for the end-of-conversion line "high"

(5) repeat steps 1-4 for 1024 readings

The 1024 values were then transferred to the host computer and plotted. The number of samples per cycle of the sinewave were counted and divided by the known period to yield a sampling frequency of 3000 samples per second.

## 5.3 Peripheral Data Transfer Rate

The maximum rate at which the microcomputer system can communicate to a peripheral system was determined by executing an infinite loop consisting of a read/write sequence to the programmable peripheral interface (PPI). The loop was repeated at a rate of 4.5 MHz. Thus the system could transfer up to 9 M-words of data per second. No external device was connected to the microcomputer system; therefore, the value does not take into account the time required for a transfer protocol to an external system.

## 5.4 Power Consumption

The power consumption of the microcomputer system was determined for the system as a whole with regard to only the 5 V supply. Since the outputs of the HP general purpose parallel interface (GPIO) are all open collector, resistive line terminators had to be installed on the microcomputer system board. The line terminators are peculiar to the host computer and are not considered part of the microcomputer system. With the line terminators installed and the system connected to the HP-9845B computer, the power consumption is approximately 2.5 W. With the line terminators installed and the system disconnected from the host computer, the power consumption is approximately 1.42 W. With the line terminators removed and the system not connected to the host computer, the power consumption is approximately 207.5 mW. These values were taken with the system executing the initialization procedure. When the AAMP is halted, the power consumption decreased by an additional 12.5 mW.

## 6. Concluding Remarks

The AAMP is a high-performance single-chip microprocessor which appears to meet all published performance claims. The AAMP's low power consumption and floating-point arithmetic capabilities make it attractive over other microprocessors such as the Motorola 68000 and National 16032 which are not available in low-power versions or the Intel 80C86 which is low powered but does not have built-in floating point arithmetic capabilities.[1]

The AAMP is designed for use in systems of medium to high complexity. The asynchronous, interlocking, handshaking bus protocol allows the AAMP to be easily interfaced to a wide variety of devices. The AAMP can be configured so that bus operations have no delay or have as long of delay as required by the devices being accessed. Because the AAMP uses a handshaking bus protocol, external hardware must generate the transfer acknowledge signal if the devices being interfaced do not provide this feature.

The only difficulty encountered in the AAMP based microcomputer design was associated with the AAMP's read/write line. The read/write line only indicates the direction of the transfer and must be combined externally with the transfer request signal to generate chip read and write enables. The read/write line is normally in the "low" (read) state. During a write cycle, the read/write line goes "high" when the bus is granted to the AAMP and returns to the "low" (read) state 5 to 10 ns before transfer request is removed. This fact is not indicated in the available timing specifications. The removal of read/write before the transfer is terminated by transfer request can have an impact on the design of the chip select circuit. For example, the initial microcomputer system design had a glitch on the PPI and RAM read enables at the end of a write cycle. The read/write line had to be delay in order to correct the problem.

Rockwell International designed the AAMP's instruction set so that the AAMP could be programmed in high-level languages such as PL/I, Jovial, and Ada. The AAMP was not intended to be programmed manually in machine code.[1] The AAMP's instruction set was found to be very structured and easy to learn. The only aspect of hand coding the AAMP which required some special attention was the calculation of jump offsets. The program counter deals with byte addresses; therefore, the actual word addresses have to be converted to byte addresses before a byte jump offset can be calculated. The procedure is not difficult but is troublesome when editing and revising programs.

The AAMP based microcomputer system's performance proved to be superior over all other microprocessors previously evaluated at Kansas State University. Using the Widrow adaptive linear predictor algorithm as a benchmark, the only processor previously evaluated which comes even close to competing with the AAMP is National Semiconductor's NSC800. When comparing the execution rate of the above algorithm using 16-bit, fixed-point arithmetic, the AAMP is 282 percent faster than the NSC800.

The microcomputer design allows the AAMP to execute programs from random-access memory with essentially no wait states. The minimum possible transfer cycle access time to the read-only-memory is 200 ns greater than the access time to the random-access memory. Thus, programs executed from read-only memory take longer. If the benchmark algorithm had been executed from ROM, the AAMP's execution rate would decrease to 277 percent greater than that of the NSC800.

The maximum operating rates for the analog-to-digital converter and the parallel interface were also measured. The maximum rate of the Datel Intersil ADC-HC12B, analog-to-digital converter, operating in a power standby configuration, is 3000 samples per second. The maximum rate that the microcomputer system can send and receive information via the Harris 82C55A, programmable peripheral interface, is 9 M words per second.

The amount of power the microcomputer system consumes from the 5 V supply is greatly increased by the resistive line terminators required for proper operation of HP-98032A interface. Since these components are not part of the hardware required to support the AAMP, they were removed and the host computer was disconnected while measuring the power consumption of the AAMP based microcomputer system. The measure power consumption was 207.5 mW with the AAMP executing the the board's operating system program. This value indeed supports the figures that Rockwell International has published.

The Advanced Architecture Microprocessor (AAMP) developed by Rockwell International should definitely have its place in the digital systems of tomorrow. High-performance, low power consumption, and versatility make the AAMP a very remarkable device.

## Acknowledgements

This research project was sponsored by the Systems Engineering Division, Organization 5238, Sandia National Laboritories, Kirtland Air Force Base, Albuquerque, New Mexico.

I would like to give a special thanks to the United States Air Force for sponsoring my degree. Thanks is also extended to Dr. D. H. Lenhert, Dr. M. S. P. Lucas, and Dr. J. G. Thompson for serving as ccmmittee members. I would also like to acknowledge Mike Kadlicek for the fine job he did in drawing the AAMP based microcomputer schematic utilizing the HP-9845 Engineering Graphics System.

## References

[1]Dave W. Best et al, "An Advanced-Architecture CMOS/SOS Microprocessor," IEEE Micro, 2:3, August 1982, pp 10-26.

[2]Kenneth L. Albin, "An Evaluation of Rockwell's Advanced Architecture Microprocessor for Digital Signal Processing Applications" (MS Thesis, Kansas State University, 1984).

[3]AAMP, CAPS-7, and CAPS-10 Instruction Set Architecture, Processor Technology Section, Advanced Technology and Engineering, Collins Avionics Division, Rockwell International, 1982.

[4]AAMP Hardware Reference Manual, Rockwell International, 1984.

[5]Kenneth J. Hass, "On a Microprocessor Implementation of an Intrusion-Detection Algorithm" (MS Report, Kansas State University, 1978), pp 17-18.

[6]Dave W. Best, "Design of an Advanced Architecture Microprocessor" (MS Thesis, Iowa State University, 1984), pp 100-114.

[7]Confirmed by Dave W. Best, Design Engineer for the Advanced Architecture Microprocessor, phone conversation, Rockwell International, April 10, 1985.

[8]HP 98032A Installation and Service Manual, Hewlett-Packard Desktop Computer Division, 1976.

# Appendix A

## CMOS vs CMOS/SOS Versions of the Advanced Architecture Microprocessor (AAMP)

The Advanced Architecture Microprocessor (AAMP), developed by Rockwell International, is currently in a transition state. The AAMP was first implemented using two-micrometer CMOS/SOS VLSI technology. Problems in producing the SOS devices has forced Rockwell International to change the AAMP over to a VLSI CMOS technology. Along with the change in technology, some features and characteristics of the AAMP have also been changed. The notable differences are described below.

The CMOS AAMP has an output enable pin ($\overline{\text{OE}}$) which the CMOS/SOS AAMP does not have. The output enable feature is complemented by a three-state operation added to the address lines and the bus control lines ($\overline{\text{R}}$/W, E/$\overline{\text{U}}$, C/$\overline{\text{D}}$, and XRQ). The CMOS AAMP will connect into multiple master bus systems without the need of external transceivers. When $\overline{\text{OE}}$ is "high", the address and control lines will be in the high impedance state which effectively removes the AAMP from the system. For applications where the CMOS AAMP is the only master, $\overline{\text{OE}}$ could be strapped to ground to continuously enable the outputs.

The bus timing on the CMOS AAMP is also changed; but the basic asynchronous, interlocking, bus protocol remains unchanged.

The CMOS AAMP has a built-in 50ns hold time after XRQ goes "low" for address, data, $\bar{R}$/W, E/$\bar{U}$, and C/$\bar{D}$. The CMOS/SOS AAMP had no hold time but was equipped with a hold line to control external hardware to accomplish this purpose. The synchronization of XAK to the 10 MHz clock has also been changed. Transfer acknowledge is synchronized to the 20 MHz clock which will reduce transfer cycle times.

The CMOS AAMP will be TTL compatible as with the CMOS/SOS version. The difference is that the CMOS AAMP will have a higher fanout. The exact fanout is yet to-be-determined.

The CMOS AAMP pin assignment has also undergone some changes. The changes only affect pin functions which have been either added or deleted. Table A-1 lists the changed pin assignments.

Overall, the changes in the AAMP architecture are improvements. The only aspect of the change which could be considered otherwise, is that the CMOS AAMP will use more power than the CMOS/SOS version. The exact power consumption is yet to-be-determined.

Table A-1

Pin Assignment Changes

| Pin Number | CMOS AAMP Function | SOS AAMP Function |
|---|---|---|
| 9 (F) | VDD | NC |
| 18 (L2) | GND | NC |
| 34 (L10) | SIN* | NC |
| 35 (K11) | NC | HLD |
| 39 (H11) | NC | HB |
| 40 (H10) | OE | LB |
| 62 (A5) | GND | NC |

*test pin

## Appendix B

## Microcomputer System Board Layout and Parts List

The component layout of the microcomputer system is given in Figure B-1. The parts list follows:

| Reference | Part Number | Description |
|---|---|---|
| C1 | UNKNOWN | 68 pF ceramic disc capacitor |
| C2 | UNKNOWN | 68 pF ceramic disc capacitor |
| C3 | UNKNOWN | 10 μF 20V tantalum capacitor |
| C4 | UNKNOWN | 0.0047 μF polystyrene capacitor |
| CR1 | MV5054-1 | LED |
| P1 | 52-1150-50 | 50-pin wire wrap edge connector |
| R1 | UNKNOWN | 1/4 W, 1K ohm carbon resistor |
| R2 | UNKNOWN | 1/4 W, 10K ohm carbon resistor |
| R3 | UNKNOWN | 1/4 W, 10K ohm carbon resistor |
| R4 | UNKNOWN | 1/4 W, 33K ohm carbon resistor |
| R5 | UNKNOWN | 1/4 W, 3.3K ohm carbon resistor |
| R6 | UNKNOWN | 1/4 W, 330 ohm carbon resistor |
| R7 | UNKNOWN | 1/4 W, 10K ohm carbon resistor |
| R8 | UNKNOWN | 1/4 W, 10K ohm carbon resistor |
| R9 | UNKNOWN | 9K ohm trimmer potentiometer |
| R10 | UNKNOWN | 10K ohm trimmer potentiometer |
| R11 | UNKNOWN | 1/4 W, 830K ohm carbon resistor |
| R12 | UNKNOWN | 100K ohm trimmer potentiometer |
| S1 | 570-15 | DIALIGHT SPDT momentary switch |
| S2 | 570-15 | DIALIGHT SPDT momentary switch |
| U1 | 74HC04N | Hex invertor |
| U2 | 74HC04N | Hex invertor |
| U3 | 74HC04N | Hex invertor |
| U4 | 74HC32N | Quad 2-input OR gate |
| U5 | 74HC32N | Ouad 2-input OR gate |
| U6 | 74HC74N | Dual D Flip Flop with PR & CLR |
| U7 | 74HC51N | Dual AND-OR_INVERT gate |
| U8 | 74HC00N | Quad 2-input NAND gate |
| U9 | 74HC00N | Quad 2-input NAND gate |
| U10 | 74HC02N | Quad 2-input NOR gate |
| U11 | 74HC138N | 3-to-8 line decoder |
| U12 | 74HC194N | 4-bit Bidir. Univ. Shift Register |
| U13 | 74HC194N | 4-bit Bidir. Univ. Shift Register |

| Reference | Part Number | Description |
|---|---|---|
| U14 | 74HC157N | Quad 2-input Multiplexer |
| U15 | HM6116P-2 | CMOS 2048x8-bit static RAM |
| U16 | HM6116P-2 | CMOS 2048x8-bit static RAM |
| U17 | CD82C55A | Programmable Peripheral Interface |
| U18 | AAMP | Advanced Architecture Microprocessor |
| U19 | NMC27C16Q-35 | CMOS 2048x8-bit UV Erasable ROM |
| U20 | NMC27C16Q-35 | CMOS 2048x8-bit UV Erasable ROM |
| U21 | CD82C55A | Programmable Peripheral Interface |
| U22 | 74HC374N | Tri-state Octal D Flip Flop |
| U23 | 74HC374N | Tri-state Octal D Flip Flop |
| U24 | 74HC74N | Dual D Flip Flop with PR & CLR |
| U25 | 74HC4050N | Hex Logic Level Down Converter |
| U26 | 74HC4050N | Hex Logic Level Down Converter |
| U27 | 74HC4050N | Hex Logic Level Down Converter |
| U28 | 74HC4040N | 12 Stage Binary Counter |
| U29 | ADC-HC12BMC | 12-bit CMOS ADC |
| U30 | AD583KD | Sample-and-Hold |
| U31 | MC14504B | Hex Logic Level Up Converter |
| U32 | 316E221331 | 220/330 ohm line terminators |
| U33 | 316E221331 | 220/330 ohm line terminators |
| XTL | 07XTL20.00HH | 20MHz crystal |
| (28 each)* | A5C103K | 0.01 uF despiking capacitor |
| (4 each) | | Banana Jacks, Double-D Body |
| (1 each) | | 6x8 bare wire wrap board |

(all integrated circuits are installed in wire wrap sockets which are not listed)

* Despiking capacitors are connected between ground and Vcc on integrated circuit chips U1-13 and U15-29.

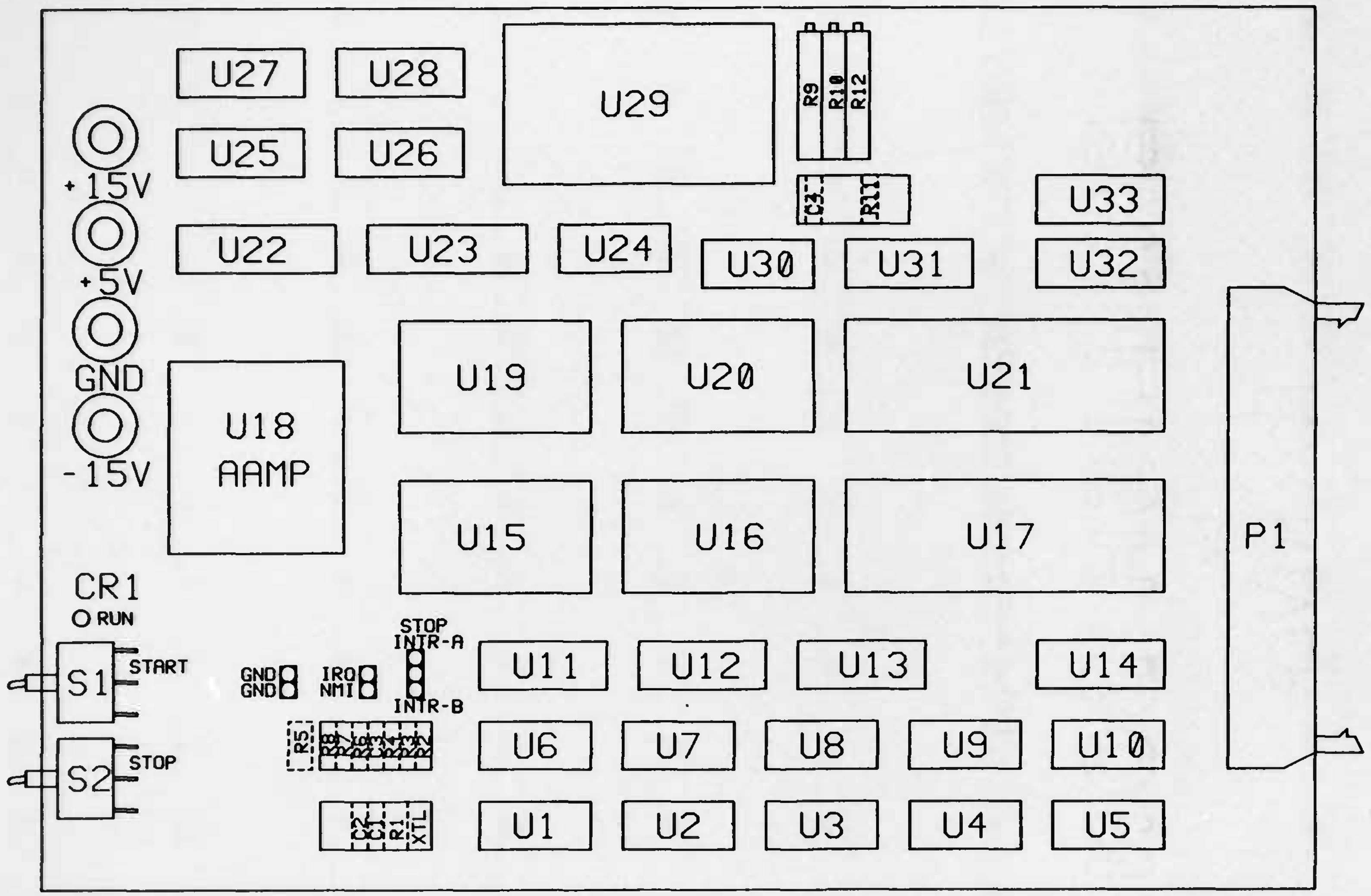


Figure B-1. Microcomputer Board Layout

# Appendix C

## Guide for Programming the CD82C55A

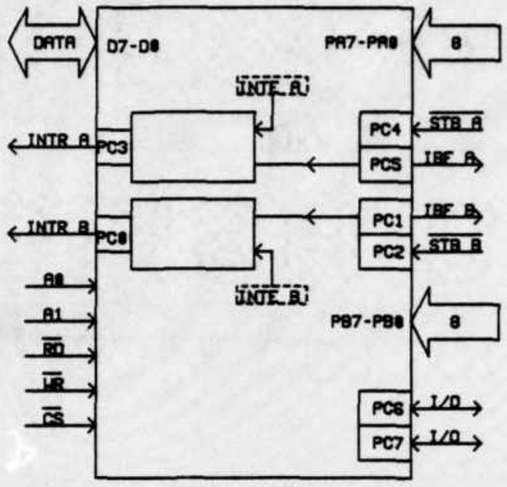
The CD82C55A, programmable peripheral interface (PPI), has three major modes of operation that can be selected by system software. These include mode 0, basic input/output; mode 1, strobed input/output; and mode 2, bidirectional bus. Mode 1 is used in the microcomputer system design and will be the only mode considered.

The PPI has three ports, two control registers, and a status register. In mode 1, Port A and Port B can be configured in four different ways. Figure C-1 shows three configurations. The fourth configuration is the opposite of Figure C-1(C). Port C is defined as shown. All registers and ports are bus addressable. Table C-1 gives the address for the ports and registers. The format of the status register is given in Figure C-2.

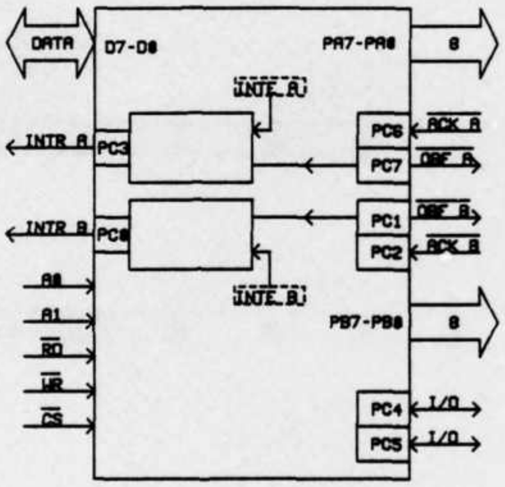
Table C-1

Port and Register Addresses

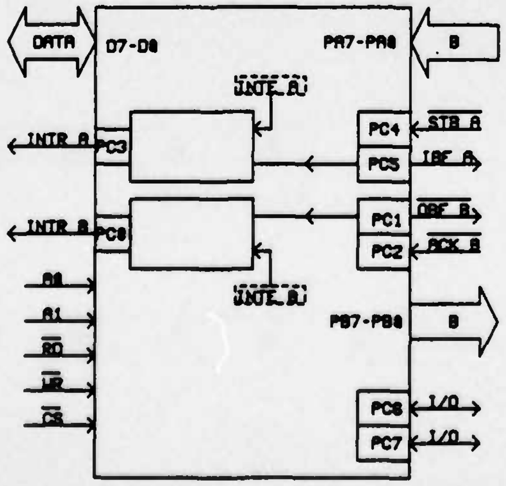
| Address | Designation |
|---|---|
| 00 | Port A |
| 01 | Port B |
| 10 | Port C (Status Register in mode 1) |
| 11 | Control Function (Control Register and Bit Set/Reset) |



A -- PORT A STROBED INPUT / PORT B STROBED INPUT



B -- PORT A STROBED OUTPUT / PORT B STROBED OUTPUT



C -- PORT A STROBED INPUT / PORT B STROBED OUTPUT

Figure C-1. PPI Mode 1 Configurations

| BIT 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| I/O | I/O | IBF A | INTE A | INTR A | INTE B | OBF B | INTR B |

(PC6 and PC7 are input/out lines)

Figure C-2. Port C (Status Register) Format

The control function address has two purposes depending on bit 7 of the byte stored to that location. If bit 7 is a "one", the byte is defined as the control word and is used to set the PPI mode, port direction, and input/output line direction as shown in Table C-2. If bit 7 is "zero", the byte is a bit set/reset command used to set interrupt enables and to control the single bit input/output lines as shown in Table C-3. Bits 1-3 define which bit is selected and bit 0 specifies the value it is to receive. The selected bit number corresponds to the status register bit and associated function. The PPI control function address can also be accessed by a read operation to return the contents of the control register.

Table C-2

Control Word Bit Assignment

| Bit | Function |
|---|---|
| 0 | Don't Care |
| 1 | Port B direction * |
| 2 | Port B mode (1 = mode 1) |
| 3 | PC6 and PC7 direction * |
| 4 | Port A direction * |
| 5 | Port A Mode bit 0 |
| 6 | Port A Mode bit 1 (01 = mode 1) |
| 7 | Function Select (1 = Active) |

(* Direction: 1 = input / 0 = output)

Table C-3

Set/Reset Function Bit Assignment

| Bit | Function |
|---|---|
| 0 | Bit Set/Reset (1 = SET / 0 = RESET) |
| 1 | Bit (0) - |
| 2 | Bit (1) --> Bit Select |
| 3 | Bit (2) - |
| 4 | Don't Care |
| 5 | Don't Care |
| 6 | Don't Care |
| 7 | Function Select (0 = Active) |

# Appendix D

## Edge Connector Pin Assignment and HP-98032A General Purpose I/O Jumper Requirements

The microcomputer system was connected to the HP-98032A general purpose parallel input/output interface (GPIO) through a 50-pin edge connector. The pin assignment for the edge connector is the same as that suggested in the HP-98032A Installation and Service Manual.[8] This appendix includes a listing of the pin assignments as well as a list of the GPIO internal jumpers that are required for the handshake between the two systems to work properly.

### Edge Connector Pin Assignment

| Pin Number | Microcomputer Connection | Microcomputer Signal Name | GPIO Signal Name |
|---|---|---|---|
| A1 | GND | GND | GND |
| A2 | U32-1; U21-37(PA7) | DI-15 | DO-15 |
| A3 | U32-2; U21-38(PA6) | DI-14 | DO-14 |
| A4 | U32-3; U21-39(PA5) | DI-13 | DO-13 |
| A5 | U32-4; U21-40(PA4) | DI-12 | DO-12 |
| A6 | U32-5; U21-1(PA3) | DI-11 | DO-11 |
| A7 | U32-6; U21-2(PA2) | DI-10 | DO-10 |
| A8 | U32-7; U21-3(PA1) | DI-09 | DO-09 |
| A9 | U32-9; U21-4(PA0) | DI-08 | DO-08 |
| A10 | U32-10; U17-37(PA7) | DI-07 | DO-07 |
| A11 | U32-11; U17-38(PA6) | DI-06 | DO-06 |
| A12 | U32-12; U17-39(PA5) | DI-05 | DO-05 |
| A13 | U32-13; U17-40(PA4) | DI-04 | DO-04 |
| A14 | U32-14; U17-1(PA3) | DI-03 | DO-03 |
| A15 | U32-15; U17-2(PA2) | DI-02 | DO-02 |
| A16 | U33-1; U17-3(PA1) | DI-01 | DO-01 |
| A17 | U33-2; U17-4(PA0) | DI-00 | DO-00 |

| Pin Number | Microcomputer Connection | Microcomputer Signal Name | GPIO Signal Name |
|---|---|---|---|
| A18 | NC | --- | GND |
| A19 | U33-3; U14-2,6,10 | STB-A,ACK-B | PCTL |
| A20 | U33-4; U14-1 | SELECT | I/O |
| A21 | NC | --- | PRESET |
| A22 | U33-5; U17-11(PC6) | I/O-1 | CTL 0 |
| A23 | U33-6 | --- | CTL 1 |
| A24 | GND | GND | GND |
| A25 | NC | --- | DRAIN |
| B1 | GND | GND | GND |
| B2 | U21-25(PB7) | DO-15 | DI-15 |
| B3 | U21-24(PB6) | DO-14 | DI-14 |
| B4 | U21-23(PB5) | DO-13 | DI-13 |
| B5 | U21-22(PB4) | DO-12 | DI-12 |
| B6 | U21-21(PB3) | DO-11 | DI-11 |
| B7 | U21-20(PB2) | DO-10 | DI-10 |
| B8 | U21-19(PB1) | DO-09 | DI-09 |
| B9 | U21-18(PB0) | DO-08 | DI-08 |
| B10 | U17-25(PB7) | DO-07 | DI-07 |
| B11 | U17-24(PB6) | DO-06 | DI-06 |
| B12 | U17-23(PB5) | DO-05 | DI-05 |
| B13 | U17-22(PB4) | DO-04 | DI-04 |
| B14 | U17-21(PB3) | DO-03 | DI-03 |
| B15 | U17-20(PB2) | DO-02 | DI-02 |
| B16 | U17-19(PB1) | DO-01 | DI-01 |
| B17 | U17-18(PB0) | DO-00 | DI-00 |
| B18 | NC | --- | DRAIN |
| B19 | U14-9 | IBF-A;PFLG | PFLG |
| B20 | NC | --- | PSTS |
| B21 | NC | --- | EIR |
| B22 | U17-15 | OBF-B | STI 0 |
| B23 | NC | --- | STI 1 |
| B24 | GND | GND | GND |
| B25 | NC | --- | NC |

## GPIO Jumper List

| Jumper Number | Status | Function |
|---|---|---|
| 1 | REMOVED | Input data negative logic |
| 2 | REMOVED | Output data negative logic |
| 3 | REMOVED | PCTL(high) = CLEAR |
| 4 | INSTALLED | PFLG(high) = READY |
| 5 | DON'T CARE | Logic of PSTS |
| 6 | DON'T CARE | Pulse vs Full Handshake |
| 7 | DON'T CARE | DMA Enable |
| 8,E | INSTALLED | Latch input when PFLG goes READY for BUSY |
| 9,D | REMOVED | Alternate data latch function |
| A,C | REMOVED | Alternate data latch function |
| B,F | INSTALLED | I/O word mode |

# Appendix E

## AAMP Based Microcomputer Operating System

The operating system developed for the AAMP based microcomputer system consists of three parts: the Executive Entry Table, the initialization procedure, and the trap handling procedure.

### Executive Entry Table

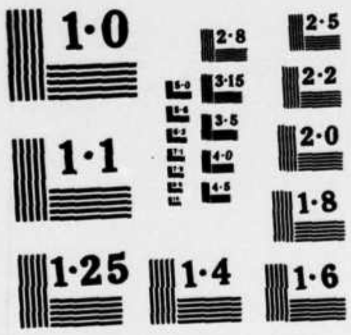
The Executive Entry Table used in the microcomputer system is as follows:

| Address | Contents | Description |
|---|---|---|
| 000000 | 0000 | Continuation Status Pointer |
| 000001 | 1701 | Initial Executive Stack Limit |
| 000002 | 17F0 | Initial Executive Top-of-Stack (TOS) |
| 000003 | 0020 | Initial Executive Procedure ID (PROCID) |
| 000004 | 0000 | Bus Error PROCID |
| 000005 | 0000 | Nonmaskable Interrupt PROCID |
| 000006 | 0000 | Maskable Interrupt PROCID |
| 000007 | 0060 | Trap PROCID |
| 000008 | 0000 | Exception PROCID |

The "0000" in the continuation status pointer defines all resets to be "cold", that is the processor must go through the initialization procedure each time the system is reset. A procedure identifier PROCID of "0000" means that procedure does not exist; therefore, if a call is made to a procedure with a zero PROCID, the processor will halt. A PROCID is the byte

AD-A158 130 THE DESIGN AND HARDWARE EVALUATION OF AN ADVANCED 16-BIT LOW-POWER HIGH P..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH G S MAUERSBERGER 1985 2/2

UNCLASSIFIED AFIT/CI/NR-85-72T F/G 9/2 NL


1·0
2·8
2·5
3·15
2·2
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6

address offset pointer to a procedure's header. The executive stack limits on the other hand are word address offsets. The Initial Executive TOS location is reserved for the User Processor State Descriptor (PSD) Pointer which specifies which user task is active.

## Initialization Procedure

The initialization procedure is always invoked when power is applied or when the system is reset because the Continuation Status Pointer is zero. The procedure performs three functions in the microcomputer system: configures the programmable peripheral interface, accepts a user program from the host computer and stores it in RAM, and defines the location of the user task. The programmable peripheral interface (PPI) is configured by the initialization program storing "BCBC" to system address "002003". The procedure can then accept user program code and data from the host computer (HP-9845B). The initialization program requires the host computer to send information in a code (or data)/address sequence. The initialization program checks the end-of-program transfer line after receiving each code/address word pair. When bit 6 of the PPI status word changes to a "one", the initialization program stores "1010" in the User PSD Pointer location. The initialization procedure then terminates with a RETURN instruction which causes the processor to context switch to the user mode and execute the user program defined by the User PSD Table located at address "001010".

The following is a listing of the microcomputer system's initialization procedure. The leading two zeros have been omitted from the addresses because all addresses are with respect to code environment zero.

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 0010 | 0000 | 0000 | | procedure header |
| 0011 | BC1A | BCBC1A | LIT16 | load control word |
| 0012 | 54BC | | | |
| 0013 | 2003 | 200354 | ASNSI | store PPI control word |
| 0014 | BC1A | BCBC1A | LIT16 | load control word |
| 0015 | 1CBC | | | |
| 0016 | 2003 | 20031C | REFSI | get PPI control word |
| 0017 | 19EB | EB | EQ | ?is control word correct |
| 0018 | EE0E | 0E19 | LIT8N | |
| | | EE | SKIPZ | jump to 0011(L) if "no" |
| 0019 | 021C | 20021C | REFSI | get PPI status |
| 001A | 1820 | 2018 | LIT8 | |
| 001B | E820 | E8 | AND | ?is input buffer full |
| 001C | 0819 | 0819 | LIT8N | |
| 001D | 1CEE | EE | SKIPZ | jump to 0019(L) if "no" |
| 001E | 2000 | 20001C | REFSI | get data from PPI |
| 001F | 021C | 20021C | REFSI | get PPI status |
| 0020 | 1820 | 2018 | LIT8 | |
| 0021 | E820 | E8 | AND | ?is input buffer full |
| 0022 | 0819 | 0819 | LIT8N | |
| 0023 | 1CEE | EE | SKIPZ | jump to 001f(L) if "no" |
| 0024 | 2000 | 20001C | REFSI | get RAM address |
| 0025 | 1CD3 | D3 | ASNS | store data at address |
| 0026 | 2002 | 20021C | REFSI | get PPI status |
| 0027 | 4018 | 4018 | LIT8 | |
| 0028 | 19E8 | E8 | AND | ?is transfer complete |
| 0029 | EF21 | 2119 | LIT8N | |
| | | EF | SKIPNZ | jump to 0019(L) if "no" |
| 002A | 101A | 10101A | LIT16 | |
| 002B | 5410 | 17F054 | ASNSI | store User PSD Pointer |
| 002C | 17F0 | 10 | LIT4A.0 | |
| 002D | 5F10 | 5F | RETURN | procedure end |

## Trap Handling Procedure

If no trap handling procedure exists, the processor will halt when an error occurs in the user mode or when the user program terminates by a RETURN instruction. A trap number corresponding to the error condition is placed on the executive stack before the trap handling procedure is invoked. The ability to recover this information is useful in determining the cause of processor termination. The trap handling program given below stores the trap number at RAM address "001000". The procedure also stores the value of the user syllable program counter (SPCR) at RAM address "001001". The trap handler then executes the HALT instruction. The trap handling procedure listing follows:

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 0030 | 0000 | 0000 | | procedure header |
| 0031 | 5400 | 00 | REFSL.0 | get trap number |
| 0032 | 1000 | 100054 | ASNSI | save trap number |
| 0033 | 141C | 10141C | REFSI | get User SPCR |
| 0034 | 5410 | | | |
| 0035 | 1001 | 100154 | ASNSI | save User SPCR |
| 0036 | 00FE | FE | HALT | procedure end |

## Appendix F

### System Status Retrieval Programs

A program scheme was developed to aid in determining the cause of premature program terminations. The trap handling routine in the operating system records the trap number and SPCR, where the trap originated, in RAM address "001000" and "001001" respectively. An exception handling program was also developed to identify program locations which cause accumulator overflows and underflows. When an overflow or underflow occurs, an exception number corresponding to the error condition is passed to the exception handler. The exception handler records the exception number and the user SPCR, corresponding to where the exception occurred, in RAM locations "001002" and "001003" respectively.

The information saved by the exception and trap handlers can be retrieved by the host computer. First, the microcomputer system is reset. Next, the host computer can send a short program to the microcomputer system which returns the saved information.

Listings for both the exception handler program and status retrieval program are given below. The leading zeros have been omitted from the addresses.

## Exception Handling Program

| Address | Contents | Opcodes | Instruction | Comments |
|---|---|---|---|---|
| 1100 | 0000 | 0000 | | procedure header |
| 1101 | 5310 | 10 | LIT4A.0 | |
| | | 53 | LOCL | |
| 1102 | E514 | 14 | LIT4A.4 | |
| | | E5 | SUB | |
| 1103 | 5455 | 55 | REFS | get SPCR from user stack mark |
| 1104 | 1003 | 100354 | ASNSI | save SPCR |
| 1105 | 5400 | 00 | REFSL.0 | get exception number |
| 1106 | 1002 | 100254 | ASNSI | save exception number |
| 1107 | 5810 | 10 | LIT4A.0 | |
| | | 58 | TRAP | trap #8 |

(Note that the User Exception PROCID must be set to "2200".)

## Status Retrieval Program

-User PSD Table:

| Address | Contents | Description |
|---|---|---|
| 1010 | 1600 | Stack Limit |
| 1011 | 1700 | Top of Stack |
| 1012 | 0000 | Local Environment Pointer |
| 1013 | 0000 | Data Environment Pointer |
| 1014 | 0000 | Syllable Program Counter |
| 1015 | 0000 | Code Environment Pointer |
| 1016 | 2040 | Task PROCID (addr. 1020) |
| 1017 | 0000 | Task Code Environment |
| 1018 | 0000 | Exception PROCID |
| 1019 | 0000 | Exception Code Environment |

-Program Listing:

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 1020 | 0002 | 0002 | | procedure header |
| 1021 | 4111 | 11 | LIT4A.1 | |
| | | 41 | ASNSL.1 | k=0 |
| 1022 | 031C | 10031C | REFSI | get exception SPCR |
| 1023 | 1C10 | | | |
| 1024 | 1002 | 10021C | REFSI | get exception number |
| 1025 | 011C | | | |
| 1026 | 1C10 | 10011C | REFSI | get trap SPCR |
| 1027 | 1000 | 10001C | REFSI | get trap number |
| 1028 | 021C | 20021C | REFSI | get PPI status |
| 1029 | 1220 | 12 | LIT4A.2 | |
| 102A | 19E8 | E8 | AND | ?is output buffer empty |

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 102B | EE07 | 0719 | LIT8N | |
| | | EE | SKIPZ | jump to 1028(L) if "no" |
| 102C | 0154 | 200154 | ASNSI | output value |
| 102D | 7A20 | 017A | INCSLE | k = k + 1 |
| 102E | 0101 | 01 | REFSL.1 | get k |
| 102F | EC14 | 14 | LIT4A.4 | |
| | | EC | GR | ?is k > 4 |
| 1030 | 1219 | 1219 | LIT8N | |
| 1031 | 12EE | EE | SKIPZ | jump to 1028(L) if "no" |
| | | 12 | LIT4A.2 | |
| 1032 | 005F | 5F | RETURN | procedure end |

## Appendix G

## Listings for the Widrow Adapter Linear Predictor Algorithm

Three versions of the Widrow adaptive linear predictor algorithm were used to benchmark the AAMP. The Ada-subset compiler resident on a VAX11-780 at the Rockwell Collins facility in Cedar Rapids, IA was used to encode the standard floating-point and standard fixed-point fractional versions of the algorithm while the modified fixed-point fractional version was hand coded by Kenneth Albin.[2]

The modified version of the algorithm was never implemented on any system before this evaluation; however, Ken Albin ran the standard versions on a AAMP test system at Rockwell Collins to verify that the code had no fatal errors. No actual data could be processed, and the Widrow weighting constants u and v were defined as "1" and "0", respectively. Also no overflow or exception handling was considered.

The following listings are basically as documented by Ken Albin; however, some minor changes have been made. First, the fixed-point version was originally encoded for integer data. It was changed into fixed-point fractional version because the large number of multiplications would cause accumulator overflow problems; and in addition, the Widrow constants u and v are fractional values. Second, a logic error was encountered in the moving average filter portion of the fixed-point versions. The values entering the moving average filter had not been divided by the filter length (16) which caused the running sum to overflow the accumulator. Third, a coding error was discovered in the loop where the input signal array and error array are updated. The limits on the looping variable (k) were inadvertently interchanged which caused one value to be stored in 15 of the 16 array locations. Fourth, some variable initialization changes and additions were made to insure the same response from the algorithm each time a given test data file was processed. Last, some code was added to facilitate the exchange of information between the microcomputer system and the host computer. These include

(1) polling of the programmable peripheral interface (PPI) input and output buffers to slow the microcomputer system down to the rate at which the HP-9845B computer can send and receive information

(2) inverting the data because information was passed between the two systems in negative logic (Negative logic was used to increases the transfer rate of information through the HP-98032A general purpose parallel interface.)

(3) scaling of the input and output data in the floating-point version

All changes have been flagged in the following listings. The "#" indicates a changed or added line essential for proper algorithm operation. The "&" indicates a changed or added line required for interfacing the system to the HP-9845B computer.

## Standard Widrow Adaptive Linear Predictor Algorithm

```
-Variables:

     k  :  integer -- loop variable
     f  :  number-system -- current input value
     g  :  number-system -- summation value
     e  :  number-system -- current error value and filter output
     q  :  number-system -- "alarm" output
     c  :  number-system -- intermediate value

-Arrays:

     b_array  :  number-system -- weight array
     f_array  :  number-system -- sample array
     e_array  :  number-system -- error array

-Constants:

     u  :  number-system  --  (0.96875 used)                       #
     v  :  number-system  --  (0.03125 used)                       #

-Algorithm:

     (variable initialization)

          q = 0                                                     #

          e_array (1..16) = 0                                       #

          f_array (1..16) = 0                                       #

          b_array (1..16) = 0                                       #

     (end initialization)

     (begin main body)

          Loop:  Wait for input buffer FULL                         &

               f = NOT ( input buffer )                             &

               g = 0

               for k = 1 to 16 loop

                    g = g + b_array(k) * f_array(k)

               end loop
```

```
            e = f - g

            c = v * e

            for k = 1 to 16 loop

                b_array(k) = u * b_array(k) + c * f_array(k)

            end loop

            e = e / 16                                              #

            q = q - e_array(16) + e

            Wait for output buffer EMPTY                            &

            output buffer = NOT (q * q)                             &

            for k = 15 to 1 loop                                    #

                e_array(k + 1) = e_array (k)

                f_array(k + 1) = f_array (k)

            end loop

            e_array(1) = e

            f_array(1) = f

        end main loop

    (end algorithm)
```

<u>Modified Widrow Adaptive Linear Predictor Algorithm</u>

```
-Variables:

    k   :  integer -- loop variable
    ptr  :  integer -- oldest array element pointer
    f   :  number-system -- current input value
    g   :  number-system -- summation value
    e   :  number-system -- current error value and filter output
    q   :  number-system -- "alarm" output
    c   :  number-system -- intermediate value

-Arrays:

    b_array  :  number-system -- weight array
    f_array  :  number-system -- sample array
    e_array  :  number-system -- er or array
```

```
-Constants:

     u  :  number-system  --  (0.96875 used)
     v  :  number-system  --  (0.03125 used)

-Algorithm:

     (variable initialization)

          q = 0                                                  #

          ptr = 1                                                #

          e_array (1..16) = 0                                    #

          f_array (1..16) = 0                                    #

          b_array (1..16) = 0                                    #

     (end initialization)

     (begin main body)

          Loop:  Wait for input buffer FULL                      &

               f = NOT ( input buffer )                          &

               g = 0

               for k = 1 to 16 loop

                    g = g + b_array(k) * f_array(k)

               end loop

               e = f - g

               c = v * e

               for k = 16 to 1 loop

                    b_array(k + 1) = u * b_array(k) +
                                                c * f_array(k)

               end loop

               b(1) = b(17)

               e = e / 16                                        #

               q = q - e(ptr) + e

               Wait for output buffer EMPTY                      &

               output buffer = NOT ( q * q)                      &
```

```
        e(ptr) = e
        f(ptr) = f
        if ptr = 16
            then ptr = 1
            else ptr = ptr + 1
        end if
    end main loop
(end algorithm)
```

<u>Standard Floating-Point Object Listing</u>

-User PSD Table:

| Address | Contents | Description |
|---|---|---|
| 1010 | 1600 | Stack Limit |
| 1011 | 1700 | Top of Stack |
| 1012 | 0000 | Local Environment Pointer |
| 1013 | 0000 | Data Environment Pointer |
| 1014 | 0000 | Syllable Program Counter Register |
| 1015 | 0000 | Code Environment Pointer |
| 1016 | 2040 | Task PROCID (addr. 1020) |
| 1017 | 0000 | Task Code Environment |
| 1018 | 0000 | Exception PROCID |
| 1019 | 0000 | Exception Code Environment |

-Program Listing:

| Address | Contents | Opcode | Instruction | Comments | |
|---|---|---|---|---|---|
| 1020 | 006E | 006E | | Procedure header | |
| 1021 | 8025 | 7800008025 | LIT32 | | # |
| 1022 | 0000 | | | | |
| 1023 | F778 | 69F7 | ASNDLE | u = 0.96875 | |
| 1024 | 2569 | 0000007C25 | LIT32 | | # |
| 1025 | 007C | | | | |
| 1026 | 0000 | | | | |
| 1027 | 6BF7 | 6BF7 | ASNDLE | v = 0.03125 | |
| 1028 | 0025 | 0000000025 | LIT32 | | # |
| 1029 | 0000 | | | | |
| 102A | C700 | C7 | ASNDL.7 | q = 0 | # |
| 102B | 5C11 | 11 | LIT4A.1 | | |
| 102C | 1E6D | 6D5C | ASNSLE | k = 1 | |
| 102D | 186D | 6D1E | REFSLE | get k | |
| 102E | EC10 | 1018 | LIT8 | | |
| 102F | 295B | EC | GR | ? k > 16 | |

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 1030 | 0025 | 295B | SKIPNZI | jump to 1044(H) if true |
| 1031 | 0000 | 0000000025 | LIT32 | # |
| 1032 | 1E00 | 6D1E | REFDLE | |
| 1033 | 176D | 17 | LIT4A.7 | |
| 1034 | 8C53 | 53 | LOCL | |
| 1035 | 0025 | 8C | ASNDX | b_array(k) = 0 |
| 1036 | 0000 | 0000000025 | LIT32 | # |
| 1037 | 1E00 | 6D1E | REFSLE | |
| 1038 | 186D | 2718 | LIT8 | |
| 1039 | 5327 | 53 | LOCL | |
| 103A | 258C | 8C | ASNDX | f_array(k) = 0 |
| 103B | 0000 | 0000000025 | LIT32 | # |
| 103C | 0000 | 6D1E | REFSLE | |
| 103D | 6D1E | 4718 | LIT8 | |
| 103E | 4718 | 53 | LOCL | |
| 103F | 8C53 | 8C | ASNDX | e_array(k) = 0 |
| 1040 | 6D1E | 6D1E | REFSLE | |
| 1041 | E411 | 11 | LIT4A.1 | |
| | | E4 | ADD | |
| 1042 | 6D5C | 6D5C | ASNSLE | k = k + 1 |
| 1043 | 2F19 | 2F19 | LIT8N | |
| 1044 | 1C59 | 59 | SKIP | jump to 102C(H) |
| 1045 | 2002 | 20021C | REFSI | get PPI status & |
| 1046 | 2018 | 2018 | LIT8 | & |
| 1047 | 19E8 | E8 | AND | ?is input buffer full? & |
| | | 0819 | LIT8N | & |
| 1048 | EE08 | EE | SKIPZ | jump to 1044(H) if "no"& |
| 1049 | 001C | 20001C | REFSI | get input data |
| 104A | F420 | F4 | NOT | invert & |
| 104B | D965 | 65 | CVTSD | convert to double |
| | | D9 | CVTDF | convert to floating |
| 104C | 8D25 | 000008D25 | LIT32 | & |
| 104D | 0000 | | | |
| 104E | 8700 | 87 | DIVF | & |
| 104F | 25C1 | C1 | ASNDL.1 | f = input / 4096 |
| 1050 | 0000 | 0000000025 | LIT32 | |
| 1051 | 0000 | C3 | ASNDL.3 | g = 0 |
| 1052 | 11C3 | 11 | LIT4A.1 | |
| 1053 | 6D5C | 6D5C | ASNSLE | k = 1 |
| 1054 | 6D1E | 6D1E | REFSLE | get k |
| 1055 | 1018 | 1018 | LIT8 | |
| 1056 | 5BEC | EC | GR | ? is k > 16 |
| | | 185B | SKIPNZI | jump to 1063(H) if true |
| 1057 | 3318 | 33 | REFDL.3 | get g |
| 1058 | 6D1E | 6D1E | REFSLE | |
| 1059 | 5317 | 17 | LIT4A.7 | |
| | | 53 | LOCL | |
| 105A | 1ED7 | D7 | REFDX | get b_array(k) |
| | | 6D1E | REFSLE | |
| 105B | 186D | 2718 | LIT8 | |
| 105C | 5327 | 53 | LOCL | |
| 105D | 86D7 | D7 | REFDX | get f_array(k) |
| | | 86 | MPYF | |

```
   Address   Contents   Opcode   Instruction        Comments

    105E      C384          84     ADDF
                            C3     ASNDL.3     g = g + b_array(k) *
                                                           f_array(k)
    105F      6D1E        6D1E     REFSLE      get k
    1060      E411          11     LIT4A.1
                            E4     ADD
    1061      6D5C        6D5C     ASNSLE      k = k + 1
    1062      1E19        1E19     LIT8N
    1063      3159          59     SKIP        jump to 1054(L)
                            31     REFDL.1     get f
    1064      8533          33     REFDL.3     get g
                            85     SUBF
    1065      11C5          C5     ASNDL.5     e = f - g
                            11     LIT4A.1
    1066      6D5C        6D5C     ASNSLE      k = 1
    1067      6D1E        6D1E     REFSLE      get k
    1068      1018        1018     LIT8
    1069      5BEC          EC     GR          ? is k > 16
                          225B     SKIPNZI     jump to 107B(H) if true
    106A      2222        6922     REFDLE      get u
    106B      1E69        6D1E     REFSLE
    106C      176D          17     LIT4A.7
    106D      D753          53     LOCL
                            D7     REFDX       get b_array(k)
    106E      2286          86     MPYF
    106F      356B        6B22     REFDLE      get v
                            35     REFDL.5     get e
    1070      1E86          86     MPYF
                          6D1E     REFSLE
    1071      186D        2718     LIT8
    1072      5327          53     LOCL
    1073      86D7          D7     REFDX       get f_array(k)
                            86     MPYF
    1074      1E84          84     ADDF
                          6D1E     REFSLE
    1075      176D          17     LIT4A.7
    1076      8C53          53     LOCL
                            8C     ASNDX       b_array(k) =
                                                   u * b_array(k) +
                                                   v * e * f_array(k)
    1077      6D1E        6D1E     REFSLE      get k
    1078      E411          11     LIT4A.1
                            E4     ADD
    1079      6D5C        6D5C     ASNSLE      k = k + 1
    107A      2819        2819     LIT8N
    107B      3559          59     SKIP        jump to 1067(L)
                            35     REFDL.5     get e                     #
    107C      8525  0000008525     LIT32                                 #
    107D      0000
    107E      8700          87     DIVF                                  #
    107F      37C5          C5     ASNDL.5     e = e / 16                #
                            37     REFDL.7     get q
    1080      6722        6722     REFDLE      get e_array(16)
```

```
Address  Contents   Opcode     Instruction        Comments

 1081     3585          85      SUBF
                        35      REFDL.5     get e
 1082     C784          84      ADDF
                        C7      ASNDL.7     q = q - e_array(16) + e
 1083     3737          37      REFDL.7     get q
                        37      REFDL.7     get q
 1084     2586          86      MPYF
 1085     008D  0000008D25      LIT32                                  &
 1086     0000
 1087     DB86          86      MPYF        q * q * 4096               &
                        DB      CVTFD       convert to double
 1088     F4DA          DA      CVTDS       convert to single
                        F4      NOT         invert                     &
 1089     021C      20021C      REFSI       get PPI status             &
 108A     1220          12      LIT4A.2                                &
 108B     19E8          E8      AND         ?is output buffer empty&
                      0719      LIT8N                                  &
 108C     EE07          EE      SKIPZ       jump to 1089(L) if "no"&
 108D     0154      200154      ASNSI       output result
 108E     2F20          2F      LIT4B.F                                #
 108F     6D5C        6D5C      ASNSLE      k = 15
 1090     6D1E        6D1E      REFSLE      get k
 1091     EB10          10      LIT4.0                                 #
                        EB      EQ          ? is k = 0                 #
 1092     215B        215B      SKIPNZI     jump to 10A3(H) if "yes"
 1093     6D1E        6D1E      REFSLE
 1094     4718        4718      LIT8
 1095     D753          53      LOCL
                        D7      REFDX       get e_array(k)
 1096     6D1E        6D1E      REFSLE
 1097     4918        4918      LIT8
 1098     8C53          53      LOCL
                        8C      ASNDX       e_array(k+1)=e_array(k)
 1099     6D1E        6D1E      REFDLE
 109A     2718        2718      LIT8
 109B     D753          53      LOCL
                        D7      REFDX       get f_array(k)
 109C     6D1E        6D1E      REFSLE
 109D     2918        2918      LIT8
 109E     8C53          53      LOCL
                        8C      ASNDX       f_array(k+1)=f_array(k)
 109F     6D1E        6D1E      REFSLE      get k
 10A0     E511          11      LIT4A.1
                        E5      SUB                                    #
 10A1     6D5C        6D5C      ASNSLE      k = k - 1
 10A2     2619        2619      LIT8N
 10A3     3559          59      SKIP        jump to 1090(L)
                        35      REFDL.5     get e
 10A4     49F7        49F7      ASNDLE      e_array(1) = e
 10A5     F731          31      REFDL.1     get f
 10A6     1929        29F7      ASNDLE      f_array(1) = f
 10A7     59C6        C619      LIT8N
                        59      SKIP        jump to 1044(H)
```

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 10A8 | 6E18 | 6E18 | LIT8 | |
| 10A9 | 005F | 5F | RETURN | procedure end |

-Local Variable Map:

| Local Environment Offset | Variable |
|---|---|
| 1 | f |
| 3 | g |
| 5 | e |
| 7 | q |
| 9 - 27 | b_array |
| 29 - 47 | f_array |
| 49 - 67 | e_array |
| 69 | u |
| 6B | v |
| 6D | k |

Standard Fixed-Point Object Listing

-User PSD Table:

| Address | Contents | Description |
|---|---|---|
| 1010 | 1600 | Stack Limit |
| 1011 | 1700 | Top of Stack |
| 1012 | 0000 | Local Environment Pointer |
| 1013 | 0000 | Data Environment Pointer |
| 1014 | 0000 | Syllable Program Counter Register |
| 1015 | 0000 | Code Environment Pointer |
| 1016 | 2040 | Task PROCID (addr. 1020) |
| 1017 | 0000 | Task Code Environment |
| 1018 | 2200 | Exception PROCID |
| 1019 | 0000 | Exception Code Environment |

-Program Listing:

| Address | Contents | Opcode | Instruction | Comments | |
|---|---|---|---|---|---|
| 1020 | 0036 | 0036 | | Procedure header | |
| 1021 | 4410 | 10 | LIT4A.0 | | # |
| | | 44 | ASNSL.4 | q = 0 | # |
| 1022 | 5C11 | 11 | LIT4A.1 | | |
| | | 355C | ASNSLE | k = 1 | |
| 1023 | 1E35 | 351E | REFSLE | get k | |
| 1024 | 1835 | 1018 | LIT8 | | |
| 1025 | EC10 | EC | GR | ? is k > 16 | |
| 1026 | 1D5B | 1D5B | SKIPNZI | jump to 1035(H) if true | |
| 1027 | 1E10 | 10 | LIT4A.0 | | # |
| | | 351E | REFSLE | | |
| 1028 | 1435 | 14 | LIT4A.4 | | |
| 1029 | A653 | 53 | LOCL | | |

```
Address   Contents   Opcode   Instruction       Comments

                         A6    ASNSX      b_array(k) = 0
  102A      1E10         10    LIT4A.0                              #
                       351E    REFSLE
  102B      1835       1418    LIT8
  102C      5314         53    LOCL
  102D      10A6         A6    ASNSX       f_array(k) = 0
                         10    LIT4A.0                              #
  102E      351E       351E    REFSLE
  102F      2418       2418    LIT8
  1030      A653         53    LOCL
                         A6    ASNSX      e_array(k) = 0
  1031      351E       351E    REFSLE     get k
  1032      E411         11    LIT4A.1
                         E4    ADD
  1033      355C       355C    ASNSLE     k = k + 1
  1034      2319       2319    LIT8N
  1035      1C59         59    SKIP       jump to 1023(H)
  1036      2002     20021C    REFSI      get PPI status               &
  1037      2018       2018    LIT8                                    &
  1038      19E8         E8    AND        ?is input buffer full?  &
                       0819    LIT8N                                   &
  1039      EE08         EE    SKIPZ      jump to 1035(H) if "no"&
  103A      001C     20001C    REFSI      get input data
  103B      F420         F4    NOT        invert                       &
  103C      1041         41    ASNSL.1    f = input
                         10    LIT4A.0
  103D      1142         42    ASNSL.2    g = 0
                         11    LIT4A.1
  103E      355C       355C    ASNSLE     k = 1
  103F      351E       351E    REFSLE     get k
  1040      1018       1018    LIT8
  1041      5BEC         EC    GR         ? is k > 16
                       185B    SKIPNZI    jump to 104E(H) if true
  1042      0218         02    REFSL.2    get g
  1043      351E       351E    REFSLE
  1044      5314         14    LIT4A.4
                         53    LOCL
  1045      1ED0         D0    REFSX      get b_array(k)
  1046      1835       351E    REFSLE
  1047      5314       1418    LIT8
                         53    LOCL
  1048      F9D0         DO    REFSX      get f_array(k)
                         F9    MPY                                     #
  1049      42E4         E4    ADD
                         42    ASNSL.2    g = g + b_array(k) *
                                                    f_array(k)
  104A      351E       351E    REFSLE     get k
  104B      E411         11    LIT4A.1
                         E4    ADD
  104C      355C       355C    ASNSLE     k = k + 1
  104D      1E19       1E19    LIT8N
  104E      0159         59    SKIP       jump to 103F(L)
                         01    REFSL.1    get f
```

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 104F | E502 | 02 | REFSL.2 | get g |
| | | E5 | SUB | |
| 1050 | 1143 | 43 | ASNSL.3 | e = f - g |
| | | 11 | LIT4A.1 | |
| 1051 | 355C | 355C | ASNSLE | k = 1 |
| 1052 | 351E | 351E | REFSLE | get k |
| 1053 | 1018 | 1018 | LIT8 | |
| 1054 | 5BEC | EC | GR | ? is k > 16 |
| 1055 | 1A24 | 245B | SKIPNZI | jump to 1067(H) if true |
| 1056 | 7C00 | 7C001A | LIT16 | u # |
| 1057 | 351E | 351E | REFSLE | |
| 1058 | 5314 | 14 | LIT4A.4 | |
| | | 53 | LOCL | |
| 1059 | F9D0 | D0 | REFSX | get b_array(k) |
| | | F9 | MPY | # |
| 105A | 001A | 04001A | LIT16 | v # |
| 105B | 0304 | 03 | REFSL.3 | get e |
| 105C | 1EF9 | F9 | MPY | # |
| 105D | 1835 | 351E | REFSLE | |
| 105E | 5314 | 1418 | LIT8 | |
| | | 53 | LOCL | |
| 105F | F9D0 | D0 | REFSX | get f_array(k) |
| | | F9 | MPY | # |
| 1060 | 1EE4 | E4 | ADD | |
| 1061 | 1435 | 351E | REFSLE | |
| | | 14 | LIT4A.4 | |
| 1062 | A653 | 53 | LOCL | |
| | | A6 | ASNSX | b_array(k) = u * b_array(k) + v * e * f_array(k) |
| 1063 | 351E | 351E | REFSLE | get k |
| 1064 | E411 | 11 | LIT4A.1 | |
| | | E4 | ADD | |
| 1065 | 355C | 355C | ASNSLE | k = k + 1 |
| 1066 | 2A19 | 2A19 | LIT8N | |
| 1067 | 0359 | 59 | SKIP | jump to 1052(L) |
| | | 03 | REFSL.3 | get e # |
| 1068 | B814 | 14 | LIT4A.4 | # |
| | | B8 | ARS | # |
| 1069 | 436A | 6A | DUP | # |
| | | 43 | ASNSL.3 | e = e / 16 # |
| 106A | 1E04 | 04 | REFSL.4 | get q |
| 106B | E534 | 341E | REFSLE | get e_array(16) |
| | | E5 | SUB | |
| 106C | 44E4 | E4 | ADD | |
| | | 44 | ASNSL.4 | q = q - e_array(16) + e |
| 106D | 0404 | 04 | REFSL.4 | get q |
| | | 04 | REFSL.4 | get q |
| 106E | F4F9 | F9 | MPY | q * q # |
| | | F4 | NOT | invert & |
| 106F | 021C | 20021C | REFSI | get PPI status & |
| 1070 | 1220 | 12 | LIT4A.2 | & |
| 1071 | 19E8 | E8 | AND | ?is output buffer empty& |

```
Address  Contents   Opcode  Instruction     Comments

 1072    EE07        0719    LIT8N                                   &
                       EE    SKIPZ        jump to 106F(L)  if "no"&
 1073    0154      200154    ASNSI        output result
 1074    2F20          2F    LIT4B.F                                 #
 1075    355C        355C    ASNSLE       k = 15
 1076    351E        351E    REFSLE       get k
 1077    EB10          10    LIT4A.0                                 #
                       EB    EQ           ? is k = 0                 #
 1078    215B        215B    SKIPNZI      jump to 1089(H)  if "yes"
 1079    351E        351E    REFSLE
 107A    2418        2418    LIT8
 107B    D053          53    LOCL
                       D0    REFSX        get e_array(k)
 107C    351E        351E    REFSLE
 107D    2518        2518    2518
 107E    A653          53    LOCL
                       A6    ASNSX        e_array(k+1)=e_array(k)
 107F    351E        351E    REFSLE
 1080    1418        1418    LIT8
 1081    D053          53    LOCL
                       D0    REFSX        get f_array(k)
 1082    351E        351E    REFSLE
 1083    1518        1518    LIT8
 1084    A653          53    LOCL
                       A6    ASNSX        f_array(k+1)=f_array(k)
 1085    351E        351E    REFSLE       get k
 1086    E511          11    LIT4A.1
                       E5    SUB                                     #
 1087    355C        355C    ASNSLE       k = k - 1
 1088    2619        2619    LIT8N
 1089    0359          59    SKIP         jump to 1076(L)
                       03    REFSL.3      get e
 108A    255C        255C    ASNSLE       e_array(1) = e
 108B    5C01          01    REFSL.1      get f
 108C    1915        155C    ASNSLE       f_array(1) = f
                     B019    LIT8N
 108D    59B0          59    SKIP         jump to 1035(H)
 108E    3618        3618    LIT8
 108F    005F          5F    RETURN       procedure end
```

-Local Variable Map:

```
 Local Environment Offset                    Variable

       1  -----------------------------  f
       2  -----------------------------  g
       3  -----------------------------  e
       4  -----------------------------  q
       5 - 14  ------------------------  b_array
       15 - 24  -----------------------  f_array
       25 - 34  -----------------------  e_array
       35  ----------------------------  k
```

## Modified Fixed-Point Object Listing

-User PSD Table:

| Address | Contents | Description |
|---|---|---|
| 1010 | 1600 | Stack Limit |
| 1011 | 1700 | Top of Stack |
| 1012 | 0000 | Local Environment Pointer |
| 1013 | 0000 | Data Environment Pointer |
| 1014 | 0000 | Syllable Program Counter Register |
| 1015 | 0000 | Code Environment Pointer |
| 1016 | 2040 | Task PROCID (addr. 1020) |
| 1017 | 0000 | Task Code Environment |
| 1018 | 0000 | Exception PROCID |
| 1019 | 0000 | Exception Code Environment |

-Program Listing:

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| 1020 | 0039 | 0039 |  | Procedure header |
| 1021 | 5C10 | 10 | LIT4A.1 | # |
| 1022 | 1138 | 385C | ASNSLE | q = 0 # |
|  |  | 11 | LIT4A.1 |  |
| 1023 | 375C | 375C | ASNSLE | ptr = 1 |
| 1024 | 5C11 | 11 | LIT4A.1 |  |
| 1025 | 1E35 | 355C | ASNSLE | k = 1 |
|  |  | 351E | REFSLE | get k |
| 1026 | 1835 | 1018 | LIT8 |  |
| 1027 | EC10 | EC | GR | ? is k > 16 |
| 1028 | 195B | 195B | SKIPNZI | jump to 1035(H) if true |
| 1029 | 1E10 | 10 | LIT4A.0 | # |
|  |  | 351E | REFSLE |  |
| 102A | 1335 | 13 | LIT4A.3 |  |
| 102B | A653 | 53 | LOCL |  |
|  |  | A6 | ASNSX | b_array(k) = 0 |
| 102C | 1E10 | 10 | LIT4A.0 | # |
|  |  | 351E | REFSLE |  |
| 102D | 1835 | 1418 | LIT8 |  |
| 102E | 5314 | 53 | LOCL |  |
| 102F | 10A6 | A6 | ASNSX | f_array(k) = 0 |
|  |  | 10 | LIT4A.0 | # |
| 1030 | 351E | 351E | REFSLE |  |
| 1031 | 2418 | 2418 | LIT8 |  |
| 1032 | A653 | 53 | LOCL |  |
|  |  | A6 | ASNSX | e_array(k) = 0 |
| 1033 | 357A | 357A | INCSLE | k = k + 1 |
| 1034 | 1F19 | 1F19 | LIT8N |  |
| 1035 | 1C59 | 59 | SKIP | jump to 1025(H) |
| 1036 | 2002 | 20021C | REFSI | get PPI status & |
| 1037 | 2018 | 2018 | LIT8 | & |
| 1038 | 19E8 | E8 | AND | ?is input buffer full? & |
| 1039 | EE08 | 0819 | LIT8N | & |
|  |  | EE | SKIPZ | jump to 1035(H) if "no"& |

| Address | Contents | Opcode | Instruction | Comments | |
|---|---|---|---|---|---|
| 103A | 001C | 20001C | REFSI | get input data | |
| 103B | F420 | F4 | NOT | invert | & |
| 103C | 1041 | 41 | ASNSL.1 | f = input data | |
| | | 10 | LIT4A.0 | | |
| 103D | 1842 | 42 | ASNSL.2 | g = 0 | |
| | | 3518 | LIT8 | | |
| 103E | 5335 | 53 | LOCL | | |
| 103F | 1811 | 11 | LIT4A.1 | | |
| | | 1018 | LIT8 | | |
| 1040 | 1110 | 11 | LIT4A.1 | | |
| 1041 | 108F | 00108F | DO | do for k = 1 to 16 then jump to 104A(H) | |
| 1042 | 0200 | 02 | REFSL.2 | get g | |
| 1043 | 351E | 351E | REFSLE | | |
| 1044 | 5313 | 13 | LIT4A.3 | | |
| | | 53 | LOCL | | |
| 1045 | 1ED0 | D0 | REFSX | get b_array(k) | |
| 1046 | 1835 | 351E | REFSLE | | |
| 1047 | 5314 | 1418 | LIT8 | | |
| | | 53 | LOCL | | |
| 1048 | F9D0 | D0 | REFSX | get f_array(k) | |
| | | F9 | MPY | | ‡ |
| 1049 | 42E4 | E4 | ADD | | |
| | | 42 | ASNSL.2 | g = g + b_array(k) * f_array(k) | |
| 104A | 019F | 9F | ENDO | end do loop | |
| | | 01 | REFSL.1 | get f | |
| 104B | E502 | 02 | REFSL.2 | get g | |
| | | E5 | SUB | | |
| 104C | 1A43 | 43 | ASNSL.3 | e = f - g | |
| 104D | 0400 | 04001A | LIT16 | v | |
| 104E | F903 | 03 | REFSL.3 | get e | |
| | | F9 | MPY | | ‡ |
| 104F | 365C | 365C | ASNSLE | c = v * e | |
| 1050 | 3518 | 3518 | LIT8 | | |
| 1051 | 1853 | 53 | LOCL | | |
| 1052 | 1110 | 1018 | LIT8 | | |
| | | 11 | LIT4A.1 | | |
| 1053 | 0019 | 0019 | LIT8N | | |
| 1054 | 198F | 00198F | DO | do for k = 16 to 1 then jump to 1062(L) | |
| 1055 | 1A00 | | | | |
| 1056 | 7C00 | 7C001A | LIT16 | u | |
| 1057 | 351E | 351E | REFSLE | | |
| 1058 | 5313 | 13 | LIT4A.3 | | |
| | | 53 | LOCL | | |
| 1059 | F9D0 | D0 | REFSX | get b_array(k) | |
| | | F9 | MPY | | ‡ |
| 105A | 361E | 361E | REFSLE | get c | |
| 105B | 351E | 351E | REFSLE | | |
| 105C | 1418 | 1418 | LIT8 | | |
| 105D | D053 | 53 | LOCL | | |
| | | D0 | REFSX | get f_array(k) | |

```
Address  Contents  Opcode   Instruction      Comments

 105E     E4F9       F9     MPY                                    #
                     E4     ADD
 105F     351E     351E     REFSLE
 1060     5314       14     LIT4A.4
                     53     LOCL
 1061     9FA6       A6     ASNSX       b_array(k+1) = u *
                                           b_array(k) + c *
                                           f_array(k)
                     9F     ENDO        end do loop
 1062     141E     141E     REFSLE      get b_array(17)
 1063     0344       44     ASNSL.4     b_array(1) = b_array(17)
                     03     REFSL.3     get e                      #
 1064     B814       14     LIT4A.4                                #
                     B8     ARS                                    #
 1065     436A       6A     DUP
                     43     ASNSL.3     e = e / 16                 #
 1066     381E     381E     REFSLE      get q
 1067     371E     371E     REFSLE
 1068     2418     2418     LIT8
 1069     D053       53     LOCL
                     D0     REFSX       get e_array(ptr)
 106A     E4E5       E5     SUB
                     E4     ADD
 106B     5C6A       6A     DUP
 106C     6A38     385C     ASNSLE      q = q - e_array(ptr) + e
                     6A     DUP
 106D     F4F9       F9     MPY         q * q                      #
                     F4     NOT         invert                     &
 106E     021C   20021C     REFSI       get PPI status             &
 106F     1220       12     LIT4A.2                                &
 1070     19E8       E8     AND         ?is output buffer empty&
                   0719     LIT8N                                  &
 1071     EE07       EE     SKIPZ       jump to 106E(L) if "no"&
 1072     0154   200154     ASNSI       output result
 1073     0320       03     REFSL.3     get e
 1074     371E     371E     REFSLE
 1075     2418     2418     LIT8
 1076     A653       53     LOCL
                     A6     ASNSX       e_array(ptr) = e
 1077     1E01       01     REFSL.1     get f
 1078     1837     371E     REFSLE
 1079     5314     1418     LIT8
                     53     LOCL
 107A     1EA6       A6     ASNSX       f_array(ptr) = f
 107B     1837     371E     REFSLE      get ptr
 107C     EB10     1018     LIT8
                     EB     EQ          ?is ptr = 16
 107D     055A     055A     SKIPZI      jump to 1080(H) if "yes"
 107E     5C11       11     LIT4A.1
 107F     1D37     375C     ASNSLE      ptr = 1
 1080     7A02     021D     SKIPI       jump to 1081(H)
 1081     1937     377A     INCSLE      ptr = ptr + 1
 1082     599A     9A19     LIT8N
```

| Address | Contents | Opcode | Instruction | Comments |
|---|---|---|---|---|
| | | 59 | SKIP | jump to 1035(H) |
| 1083 | 3918 | 3918 | LIT8 | |
| 1084 | 005F | 5F | RETURN | end procedure |

-Local Variable Map:

| Local Environment Offset | Variable |
|---|---|
| 1 | f |
| 2 | g |
| 3 | e |
| 4 - 14 | b_array |
| 15 - 24 | f_array |
| 25 - 34 | e_array |
| 35 | k |
| 36 | c |
| 37 | ptr |
| 38 | q |

```
1057   3318        33    REFDL.3   get g
1058   6D1E      6D1E    REFSLE
1059   5317        17    LIT4A.7
                   53    LOCL
105A   1ED7        D7    REFDX     get b_array(k)
                 6D1E    REFSLE
105B   186D      2718    LIT8
105C   5327        53    LOCL
105D   86D7        D7    REFDX     get f_array(k)
                   86    MPYF
```

```
1079   6D5C        6D5C   ASNSLE
107A   2819        2819   LIT8N
107B   3559          59   SKIP      jump to 1067(L)
                     35   REFDL.5   get e                  #
107C   8525  0000008525   LIT32                            #
107D   0000
107E   8700          87   DIVF                             #
107F   37C5          C5   ASNDL.5   e = e / 16             #
                     37   REFDL.7   get q
1080   6722        6722   REFDLE    get e_array(16)
```

```
10A1    6D5C        6D5C    ASNSLE      k = k - 1
10A2    2619        2619    LIT8N
10A3    3559          59    SKIP        jump to 1090(L)
                      35    REFDL.5     get e
10A4    49F7        49F7    ASNDLE      e_array(1) = e
10A5    F731          31    REFDL.1     get f
10A6    1929        29F7    ASNDLE      f_array(1) = f
10A7    59C6        C619    LIT8N
                      59    SKIP        jump to 1044(H)
```

```
1023   1E35       351E    REFSLE      get k
1024   1835       1018    LIT8
1025   EC10         EC    GR          ? is k > 16
1026   1D5B       1D5B    SKIPNZI     jump to 1035(H) if true
1027   1E10         10    LIT4A.0                             #
                  351E    REFSLE
1028   1435         14    LIT4A.4
1029   A653         53    LOCL
```